Universidad
Carlos III de Madrid

Departamento de Informática

PROYECTO FIN DE CARRERA

CONCURSO NIST

ANÁLISIS DEL CONCURSO (2007-2012)

AUTORA: Marian Martín Sierra

TUTORA: Lorena González Manzano

Leganés, 18 de Julio de 2014

Título: CONCURSO NIST. ANÁLISIS DEL CONCURSO (2007-2012)

Autor: MARIAN MARTÍN SIERRA

Tutora: Lorena González Manzano

EL TRIBUNAL

Presidente: José María de Fuentes

Vocal: Rafael Sotomayor

Secretario: Guillermo Suárez-Tangil

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 18 de Julio de 2014 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

A mi MADRE,

ésta era tu gran ilusión y estés donde estés

lo estarás disfrutando y estarás orgullosa de mi. Siempre te querré.

Y a Angela!!

Por fin,¡¡ LO HEMOS CONSEGUIDO!!

RESUMEN

El objetivo de este proyecto es dar a conocer el concurso SHA-3, concurso que ha transcurrido durante los últimos años y que ha promovido el Instituto Nacional de Estándares y Tecnología, también conocido como NIST, con el objetivo de encontrar el nuevo algoritmo criptográfico de función resumen SHA-3 que se utilizará de estándar de aquí en adelante.

Con este fin se ha realizado un estudio sobre criptografía en general y sobre algunos de los algoritmos criptográficos de función resumen existentes hasta el momento, además de un análisis detallado del concurso, de sus fases y de sus finalistas.

Además se ha hecho un breve resumen de los problemas existentes en la actualidad referentes a seguridad y de los escándalos concernientes a ello de todos los tiempos en la sociedad norteamericana.

INDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN Y OBJETIVOS

1.1 INTRODUCCIÓN

La definición de criptografía, palabra que viene del griego kryptos (ocultar) y grafos (escribir), viene determinada como la ciencia que mediante el tratamiento de la información la protege de modificaciones y de su utilización no autorizada, utilizando para ello técnicas y algoritmos matemáticos complejos mediante los cuales se transforman los datos para ocultar su significado, garantizando su integridad, estableciendo su autenticidad y previniendo de su repudio.

Como podemos ver, la seguridad ha sido uno de los bienes más apreciados de todos los tiempos, pero es hoy en día cuando su necesidad y su importancia se hacen más notables dados los nuevos avances en materia electrónica y digital. Hoy en día gracias a los nuevos desarrollos e implementaciones es posible hacer casi de todo con un ordenador y una buena conexión a Internet, desde mirar datos bancarios, realizar gestiones con los organismos estatales y gubernamentales, o incluso firmar hipotecas.

Es por esto que se hace necesario estar seguros, es decir, tener la certeza de que los datos personales que confiamos a terceros no puedan ser interceptados ni modificados por intermediarios. Con este fin gracias a unos algoritmos preestablecidos es posible cifrar y ocultar nuestros datos ante posibles maleantes, evitando así que estos

últimos puedan utilizar los datos para fines distintos a los que estaban destinados en un principio o en su propio beneficio.

1.2 OBJETIVO DEL PROYECTO

En este apartado se va a exponer el objetivo principal del proyecto, que a grandes rasgos es dar a conocer el concurso SHA-3 y algunas facetas de la criptografía proporcionando al lector un documento sencillo donde actualizar sus conocimientos.

1.2.1 OBJETIVOS PRINCIPALES

En 1995 la NSA (Agencia de Seguridad Nacional) junto con el NIST (Instituto Nacional de Estándares y Tecnología) desarrollaron y publicaron el algoritmo SHA-1, algoritmo de resumen estándar utilizado a nivel mundial para ocultar datos personales o de dominito personal. En el año 2004 se encontraron debilidades sobre su estructura y en el año 2005 el NIST decidió modificar su categoría de estándar y reemplazarlo creando para ello un concurso cuyo objetivo era el de encontrar un nuevo algoritmo de resumen más potente y que sirviera de estándar. Este algoritmo tendría como nombre SHA-3.

En este concurso se presentarían las candidaturas de varios grupos y de entre ellas, mediante la selección de los mejores algoritmos durante sucesivas rondas eliminatorias, se elegiría el algoritmo ganador y que sería el algoritmo utilizado de estándar a escala mundial.

Mediante una investigación, se determinó que no existía un documento fácil de leer en el que se recopilase toda la información que había hasta el momento sobre este concurso, de manera que alguien que se quisiera informar sobre él tuviera un documento de cabecera mediante el cual guiarse en su análisis, y es aquí donde se decidió realizar dicho documento y complementarlo con otro documento que proporcione al lector las nociones básicas sobre seguridad necesarias a la hora de interpretar los datos y los términos del documento, de una manera rápida y sencilla, obtener los datos más

importantes sobre los diferentes algoritmos presentados al concurso y sobre sus diferentes fases.

Por tanto el objetivo principal del proyecto es dar a conocer al lector el concurso del NIST SHA-3.

1.3 FASES DE DESARROLLO

Para la realización del proyecto, ha sido necesaria una profunda investigación sobre el concurso SHA-3 y sobre los términos más relevante de las materias seguridad y criptografía.

Fueron necesarias varias semanas previas al inicio de la ejecución del proyecto para determinar el alcance del mismo y para lograr comprender los términos sobre dichas materias.

El concurso SHA-3 ha ido pasando por distintas fases de evolución incluyendo su finalización. Esto obligó a la modificación de partes del proyecto y al aumento de algunos epígrafes dados los nuevos acontecimientos.

Posteriormente, tras los cambios realizados sobre la función ganadora resultante Keccak, se han añadido algunos puntos de interés.

Además dadas las revelaciones que Edward Snowden hizo en 2013, se ha hecho un breve análisis sobre la situación que se está viviendo desde ese momento en cuanto a seguridad se refiere.

1.4 ESTRUCTURA DE LA MEMORIA

Para facilitar la lectura de la memoria se incluye a continuación un breve resumen de cada capítulo:

CAPÍTULO 1: Introducción y Objetivos

En este capítulo se expone el problema a tratar, es decir, la no existencia de un manual sencillo donde acudir para hacer un primer análisis del concurso SHA-3. Todo esto es estructurado mediante una introducción donde se detalla la temática del proyecto, además de un apartado donde se explican las distintas fases de desarrollo de proyecto y de este apartado donde se hace un breve resumen de cada capítulo.

CAPÍTULO 2: Funciones Hash

En este capítulo se desarrolla el significado de una función hash y sobretodo se desarrollan el tipo de funciones hash criptográficas.

Se analizan las propiedades de estas funciones así como sus tipos. Además se incluye un resumen de los tipos de ataques que se pueden hacer sobre ellas y de los usos que se les pueden dar.

CAPÍTULO 3: Algoritmos Criptográficos de Función Resumen SHA Y MD5

En este capítulo se analizan los algoritmos de resumen SHA con sus variantes, SHA-1 y SHA-2 y el algoritmo MD5.

Se hace un análisis completo de sus estructuras, así como de sus antecedentes y de sus diferencias. Además para poder entender bien sus estructuras se analiza la construcción Merkle-Damgard.

CAPÍTULO 4: SHA-3

En este capítulo se desarrollan todos los aspectos y etapas del concurso SHA-3. Los antecedentes que originaron el concurso, un resumen del concurso y de las fases del mismo.

Además de comentar los 14 candidatos de la segunda fase se hace un análisis de los cinco últimos algoritmos participantes en la tercera fase del concurso.

Por otro lado se cuentan las opiniones de los seguidores del concurso así como las opiniones de criptoanalistas en general y de criptoanalistas de renombre.

CAPÍTULO 5: ESTADO ACTUAL DE LA FUNCIÓN GANADORA: FIPS202

En este capítulo se hace un análisis de los cambios que se han producido en la función ganadora del concurso SHA-3 Keccak desde su elección hasta el momento además de su próxima incorporación al FIPS202 y de las dudas causadas a raíz de los documentos revelados por Edward Snowden.

CAPÍTULO 6: SEGURIDAD EN LA ACTUALIDAD, CASO SNOWDEN

En este capítulo se realiza un resumen del caso Snowden, documentos revelados, empresas implicadas, afectados, programas de espionaje utilizados y alianzas existentes.

Además se analizan los grandes informadores de todos los tiempos en Estados Unidos y los programas de espionaje utilizados anteriormente.

CAPÍTULO 7: PRESUPUESTO

En este capítulo se realiza un presupuesto de lo que ha costado la realización del proyecto atendiendo a los gastos de personal, gastos de equipos, costes directos e indirectos. Además se hace una estimación del riesgo y de los beneficios generados.

CAPÍTULO 8: GLOSARIO

En este capítulo se describen algunos de los términos y siglas utilizados en la memoria para una correcta lectura y entendimiento de ésta.

CAPÍTULO 9: CONCLUSIONES

En este capítulo se analiza del proceso que se ha seguido en la en la elaboración de este proyecto, incluyendo factores personales que han hecho que el concurso transcurriera por sus distintas fases. Además se analizan posibles trabajos futuros que lo complementaran.

CAPÍTULO 10: REFERENCIAS

En este capítulo se plasman la documentación y los artículos vía Web, en su mayoría, consultados para la elaboración de este proyecto.

CAPÍTULO 11: ANEXO: COMPARATIVAS DE LOS 5 MODELOS ELEGIDOS

En este capítulo se adjunta una comparativa de rendimiento de los 5 algoritmos finalistas que se ha realizado durante toda la ejecución de este.

Esta comparativa está realizada analizando el rendimiento sobre hardware utilizando FPGAs e implementaciones de alta velocidad ASIC.

CAPÍTULO 2

FUNCIÓNES HASH

2.1 ¿QUÉ ES UNA FUNCIÓN HASH?

El término *función hash* fue utilizado por primer vez por Hans Peter Luhn de IBM en un memorando de 1953 y posteriormente por Robert Morris en un documento de estudio más formal, para denotar al algoritmo matemático utilizado para resumir información y que dado un valor inicial nos proporciona como resultado un conjunto valor de tamaño menor.

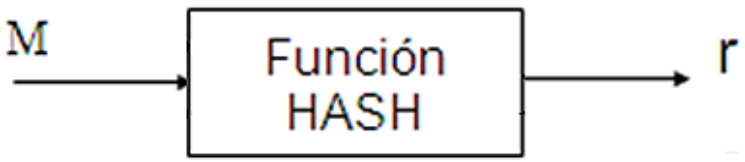


Figura 1. Diseño Función Hash

Este tipo de funciones están diseñadas bajo los criterios de las funciones de compresión las cuales a partir del procesado de un bloque de información Mk de longitud (m) producen un bloque ri de longitud (n) que se utiliza de entrada en la siguiente iteración. El resultado final será el bloque de mensaje resultante de ir encadenando recursivamente todos los resultados de las operaciones anteriores con los bloques de información:

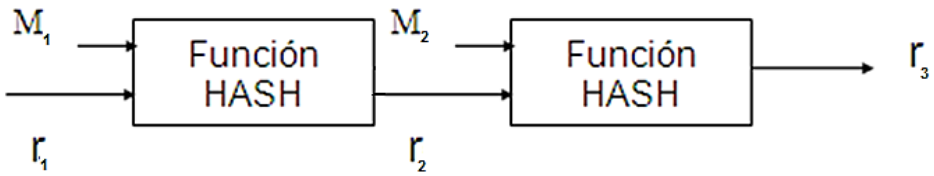


Figura 2. Diseño Función de Compresión

2.1.1 PROPIEDADES DE LAS FUNCIONES HASH

Las funciones hash y sus resultados deben poseer una serie de propiedades para su óptima utilización:

- Bajo coste: debe ser computacionalmente sencillo encontrar el resumen, es decir, dado el mensaje de entrada y mediante la utilización de un ordenador debe ser computacionalmente rápido calcular el resultado.
- Compresión: a pesar de tener un dominio muy extenso una función hash debe ser capaz de comprimir los datos en una cadena de longitud inferior.
- Uniformidad: para una clave seleccionada aleatoriamente debe haber el mismo número de posibilidades de tener un valor de hash determinado independientemente de otros elementos.
- De rango variable: propiedad muy importante siendo que en algunas funciones hash el rango de valores hash puede variar a lo largo del tiempo.
- Inyectividad: la función hash no debe tener colisiones, para cada dato de entrada se deberá producir un único valor de hash que será distinto de todos los demás. Si la función hash posee esta propiedad se dice que es una función hash perfecta.
- Determinista: una cadena de entrada siempre debe devolver el mismo valor de hash. Las funciones que no poseen esta propiedad suelen depender de parámetros externos como puede ser la fecha.

2.1.2 COLISIONES Y RESISTENCIAS

Además de todas las propiedades enumeradas anteriormente, las funciones hash deben ofrecer resistencia ante colisiones.

Una colisión se produce cuando dos mensajes distintos se introducen en una función hash y producen la misma salida, es decir, el mismo resumen.

Dependiendo de la resistencia que ofrecen las funciones hash se pueden clasificar de la siguiente manera:

1. Resistentes frente a preimágenes o primeras imágenes: una función hash es resistente a preimágenes si dado su resumen o imagen (r) es computacionalmente imposible encontrar algún mensaje (m) que cumpla h(m)=r. La complejidad asociada a este ataque es O(2n), siendo n el número de bits del hash. A las funciones que presentan este tipo de resistencia se les llama "funciones hash de un solo sentido" OWHF (One-Way Hash Function).

2. Resistentes frente a segundas preimágenes: una función hash es resistente a segundas preimágenes si dado un mensaje (m) es computacionalmente imposible encontrar otro mensaje m' distinto que cumpla h(m)=h(m'). La complejidad asociada a este ataque es de O(2n), siendo n el número de bits del hash. A las funciones que cumplen este tipo de resistencia se les llama "Weak One Way Hash Function". Otra manera de denotar este tipo de funciones es decir que son funciones resistentes a colisiones suaves.

3. Resistentes frente a colisiones: una función hash es resistente a colisiones si es computacionalmente imposible encontrar un par de mensajes distintos, m y m', que cumplan h(m)=h(m'). La complejidad asociada a este tipo de ataque es de O(2n/2), siendo n el número de bits del hash. A las funciones

hash que cumplen esta propiedad se les conoce como “Colision Resistant Hash Function” (CRHF). Este tipo de funciones además deben ser resistentes a segundas preimágenes.

4. Resistentes a pseudocolisiones o casi colisiones: una función hash es resistente a pseudocolisiones si es computacionalmente imposible encontrar dos mensajes distintos, m y m’, que cumplan h(m)=h(m’), es decir, que sus imágenes sean iguales o difieran solo en algunos bits.

5. Resistentes a pseudoimágenes o preimágenes parciales: una función hash es resistente a pseudoimágenes si dada parte de su imagen (r) es computacionalmente imposible encontrar algún mensaje (m) y algún vector de inicialización que cumpla h(m)=r.

Las resistencias más importantes son las tres primeras y la seguridad ideal viene asociada a las funciones que las poseen. Un resumen sobre la seguridad asociada a los distintos tipos de resistencias se muestra en la siguiente tabla:

PROPIEDAD	**IDEAL DE SEGURIDAD**
Resistencia a preimágenes (OWHF)	2^{n-1}
Resistencia segundas preimágenes	2^{n-1}
Resistencia a colisiones (CRHF)	$1.2 \cdot 2^{n/2}$

Tabla 1. Resumen Ideal de Seguridad [Ref.07]

Dado que hay un número finito de mensajes que podemos cifrar y por consecuente un número finito de resúmenes que podemos obtener, la presencia de colisiones es normal y todas las funciones hash las presentan.

El problema viene dado en la facilidad con que se pueden encontrar. Las colisiones pueden ser colisiones simples, es computacionalmente imposible que conocido el mensaje se pueda encontrar otro mensaje que tenga el mismo resumen, o colisiones fuertes, es computacionalmente difícil encontrar un par de mensajes que tenga el mismo resumen.

Podemos dividir las colisiones en varios grupos dependiendo de su gravedad:

- Si se tienen dos mensajes con el mismo resumen pero no se dispone de un método para encontrar más mensajes: esta situación no es grave dado que se conocen los mensajes y se puede actuar en consecuencia.
- Si se dispone de un algoritmo que pueda generar dos mensajes con el mismo resumen: esta situación no es grave puesto que los mensajes que se generarán carecerán de sentido.
- Si se tiene un mensaje y se puede calcular otro mensaje con el mismo resumen: esta situación es grave dado que dicho mensaje se podría utilizar para sustituir el mensaje original y aun careciendo de sentido, hacerlo pasar por auténtico. Esta situación sería perjudicial para redes p2p que utilizan archivos de descarga puesto que las podrían intoxicar o negar la autoría de una firma digital.
- Si se tiene un mensaje y podemos calcular otro con el mismo resumen y que posea una estructura determinada: esta situación es la más grave de todas dado que al tener una estructura determinada, el mensaje calculado sí tendá sentido.

Cuando una colisión es simple se dice que la resistencia es débil, y cuando una colisión es fuerte se dice que existe una fuerte resistencia ante las colisiones.

2.2 FUNCIÓN HASH CRIPTOGRÁFICA

Una función hash criptográfica es una función hash pública que se encarga del análisis del mensaje independientemente de su longitud y que actúa sobre un hash de longitud fija. Como añadido a las funciones de hash, este tipo de funciones cumplen una serie de propiedades que las hacen adecuadas para su utilización en criptografía.

Su origen lo podemos ubicar en 1976 cuando Diffie y Hellman presentaron el primer modelo de cifrado asimétrico de la época consistente en una clave de cifrado y una clave de descifrado distintas. En la época esto fue una novedad dado que los sistemas de cifra utilizados hasta el momento eran de clave simétrica.

2.2.1 PROPIEDADES DE LAS FUNCIONES HASH CRIPTOGRÁFICAS

Las funciones hash para que puedan ser consideradas criptográficas deben cumplir una serie de propiedades que las hagan resistentes frente ataques y deben poseer una serie de cualidades determinadas por el sistema en el que vayan a utilizar y por nivel de seguridad que se desee alcanzar.

Existe una serie de características comunes a todas ellas y que independientemente del sistema en el que se utilicen deben poseer. Estas propiedades son el determinismo, el bajo coste, la uniformidad y el efecto avalancha.

Las cualidades opcionales pero deseables son:

- Unidireccionalidad: es imposible reconstruir el mensaje inicial a partir del resumen.

- Compresión: sea cual sea la longitud del texto inicial la longitud de su resumen será siempre la misma, es decir, si se define un hash de 128 bits el hash siempre será de 128 bits con independencia de la longitud del mensaje de entrada.
- Facilidad de cálculo: dado el mensaje de entrada la obtención del resumen debe ser rápida mediante la utilización ordenador.
- Para cada entrada, la función generará una única salida, es decir, dos mensajes de entrada M y M' no pueden producir nunca la misma salida R.
- La no presencia de colisiones: Como hemos dicho anteriormente, la no presencia de colisiones es imposible que se cumpla al 100% dado que existe un número finito de mensajes que podemos cifrar y por tanto un número finito de resúmenes que podemos obtener, pero la fortaleza de una función hash radicará en la presencia del mínimo número posible de colisiones y de su dificultad para encontrarlas. Una recomendación a tener en cuenta es la longitud del resumen que se quiera obtener lo cual será prioritario a la hora de establecer la seguridad deseada. Como recomendación general se ha estipulado que las firmas deben ser de al menos 128 bits.
- Difusión o propagación de cambios: al ser el resumen una función compleja de todos los bits del mensaje, la modificación de uno de los bits del mensaje se debería propagar al resumen cambiando aproximadamente la mitad de sus bits.

2.2.2 TIPOS DE FUNCIONES HASH CRIPTOGRÁFICAS

Podemos dividir las funciones hash criptográficas en dos grandes grupos principales dependiendo de su objetivo principal.

Por un lado están los "códigos de detección de modificaciones" o MDC (Modification Detection Codes), que son funciones hash criptográficas utilizadas para la verificación de la integridad. Este tipo de funciones se subdividen en los grupos:

OWHF, funciones hash de una sola vía, y en CRHF, funciones hash resistentes a colisiones.

Por otro lado, están las funciones llamadas “códigos de autenticación de mensajes” o MAC (Message Authenticacion Codes), cuyo objetivo principal es la autenticación del origen del mensaje.

Ambos grupos de funciones tienen similitudes permitiendo ambos la autenticación de mensajes.

En el siguiente esquema se muestra la clasificación:

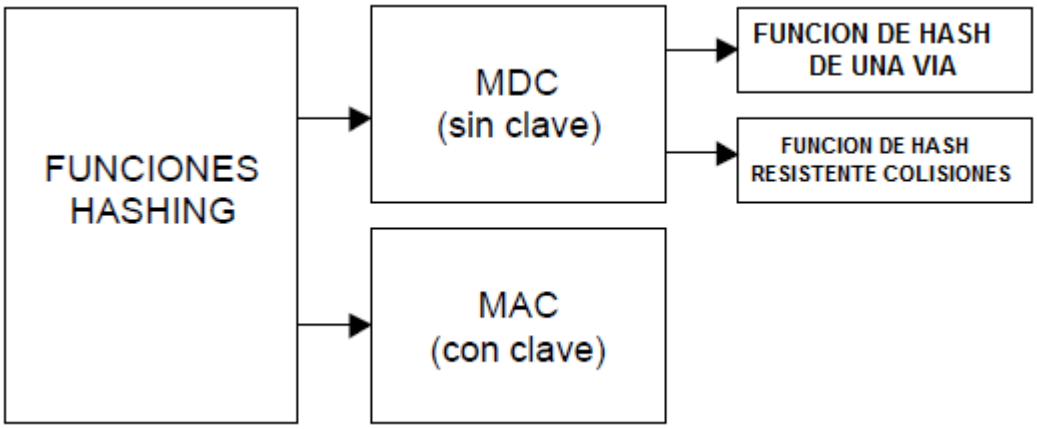


Figura 3. Clasificación de las Funciones Según Resistencia ante Ataques Criptoanalíticos

Se requerirá de un determinado nivel de resistencia ante las colisiones dependiendo del tipo de aplicación en la que se desee utilizar la función hash criptográfica:

Tipo de Resistencia requerida para las aplicaciones siguientes	Preimagen	Segunda Preimagen	Colisiones
MDC + firma digital	REQUIERE	REQUIERE	REQUIERE
MDC + canal seguro		REQUIERE	REQUIERE
MDC + BD contraseñas	REQUIERE		
MDC + clave secreta	REQUIERE	REQUIERE	REQUIERE
MDC + clave pública		REQUIERE	

Tabla 2. Requerimiento de tipo de Resistencia según tipo de Aplicación

2.2.2.1 MDC (Modification Detection Codes)

Un Código de Detección de Modificaciones es una función hash criptográfica sin clave secreta que se utiliza para la detección de modificaciones en los mensajes y que por tanto proteger la integridad de la información.

Las propiedades que deben poseer las funciones MDC para que sean fuertes frente ataque maliciosos pueden variar dependiendo de si se desea que el MDC sea OWHF, función fácil de calcular para cada entrada pero difícil de invertir dada la imagen de una entrada al azar, o que sea CRHF, función para la que es difícil encontrar dos entradas diferentes que posean el mismo valor de hash.

Por otro lado, los códigos de detección de modificaciones pueden estar construidos de tres maneras diferentes dependiendo los algoritmos utilizados para su implementación:

- Construidos mediante aritmética modular: estas funciones hash criptográficas basan su estructura en problemas de aritmética modular haciendo fácil el cambio

de los valores de referencia pero también ralentizándolos. Un ejemplo de esto es el algoritmo MASH-1.

- Construidos mediante cifradores de bloque: estas funciones hash criptográficas basan su estructura en cifradores de bloque consiguiendo así un diseño y una implementación fáciles aunque algo lentos.
- Como ejemplo de algunos cifradores de bloque podemos destacar DES (Data Encryption Standard) y AES (Advanced Encryption Standard).
- Funciones hash dedicadas: son funciones hash criptográficas que basan su estructura en códigos de detección de modificaciones siendo concebidas desde un principio para tener un rendimiento optimizado. Este tipo de funciones suele basar su construcción en la construcción Merkle-Damgard. Algunos ejemplos de funciones hash dedicadas son MD4, MD5, SHA-1, SHA2, RIMPEMD, HAIFA Y WHIRLPOOL.

FUNCIONES HASH DEDICADAS

Adentrándonos un poco más en las funciones hash dedicadas, algunos de los algoritmos más importantes utilizados a la hora de hacer los resúmenes son los siguientes:

Por un lado tenemos el algoritmo MD5, creado por Ron Rivest en 1992.

Es una mejora del algoritmo MD4 y del algoritmo MD2 y a pesar de no ser un algoritmo muy rápido es un algoritmo muy importante dado que su fortaleza reside en su seguridad y en la posibilidad de proporcionar un resumen de 128 bits. Este algoritmo está fuera del mercado desde 2005 pero desde su creación fue muy utilizado dadas su sencillez y su generalidad.

Por otro lado encontramos el algoritmo SHA-1, algoritmo propiedad del NIST (National Institute of Standards and Technology), y todas sus variantes. SHA-1 trata

con bloques de 512 y consta de 80 rondas, y aun siendo parecido a MD5 su complejidad es mucho mayor.

Otros algoritmos importantes a destacar son:

- RIPEMD: desarrollado en 1992 por la Comunidad Europea (RACE). Ofrece un resumen de 160 bits.
- N-HASH: desarrollado en 1990 por Nippon Telephone and Telegraph. Ofrece resúmenes de 128 bits.
- SNEFRU: desarrollado en 1990 por Ralph Merkle. Ofrece resúmenes de entre 128 y 256 bits pero es demostradamente lento.
- TIGER: desarrollado en 1996 por Ross Anderson y Eli Biham. Ofrece resúmenes de hasta 192 bits y además esta optimizado para máquinas Alpha de 64 bits.
- PANAMA: Desarrollado en 1998 por John Daemen y Craig Clapp. Ofrece resúmenes de 256 bits y se puede utilizar como cifrador de flujo o como función hash.
- HAVAL: desarrollado en 1192 por Yuliang Zheng, Josef Pieprzyk y Jennifer Seberry. Admite 15 configuraciones diferentes en la obtención de resúmenes de hasta 256 bits.

2.2.2.2 MAC (Message Authenticacion Codes)

Los códigos o funciones MAC son funciones hash con clave que se utilizan para comprobar la integridad y la autenticidad de la información en plataformas de IP seguras y pudiendo descubrir así ataques por parte de intermediarios.

Se calculan mediante la utilización de una función hash criptográfica con clave secreta K solo conocida por el remitente y el destinatario. En este tipo de construcción a menos que el adversario consiguiera la clave k no se podrá evaluar la función por sí

misma. Cabe destacar que ninguno de los mensajes intercambiados ayudaran al intruso a descifrar esta K más rápidamente que por búsqueda exhaustiva.

El resultado es un MAC de longitud fija con la siguiente estructura:

$$\mathbf{MAC = C_K(M)}$$

Donde M es un mensaje de longitud arbitraria y CK es una función que lo transforma en un MAC de longitud fija utilizando como parámetro la clave secreta K.

Si el MAC enviado coincide con el valor que el destinatario calcula se puede determinar que el mensaje no ha sido alterado y por tanto el remitente queda autenticado.

Existen tres tipos de funciones MAC:

- CBC-MAC: se intenta convertir un algoritmo de cifrado simétrico en una función MAC cifrando para ello el mensaje y desechando todo el texto cifrado menos el último bloque.
- HMAC: funciones MAC que, usando los hashes producidos en pasos anteriores y una clave secreta, autentican a dos usuarios mediante sistemas de clave secreta.
- UMAC: funciones que parten de la idea de que en el ataque es necesario interactuar con el sistema para comprobar si el resultado MAC que se ha generado es válido o no.

HMAC(Hash Message Authenticacion Codes)

Los códigos MAC como hemos dicho anteriormente se utilizan para autenticar dos usuarios pero para poder hacerlo directamente ambos necesitan una clave privada.

Es aquí donde aparecen las funciones HMAC que son funciones hash criptográficas que utilizando los hashes producidos en pasos anteriores y una clave secreta permiten autenticar dos usuarios mediante sistemas de clave secreta.

Se puede derivar que las funciones HMAC son una de las utilidades más importantes de los códigos MAC.

Su estructura está basada en los algoritmos MD5 o SHA-1, siguiendo la estructura:

$$\mathrm{HMAC}_k(M) = H(k \oplus c_1 || H(k \oplus c_2 || M)).$$

Para su obtención se siguen los siguientes pasos:

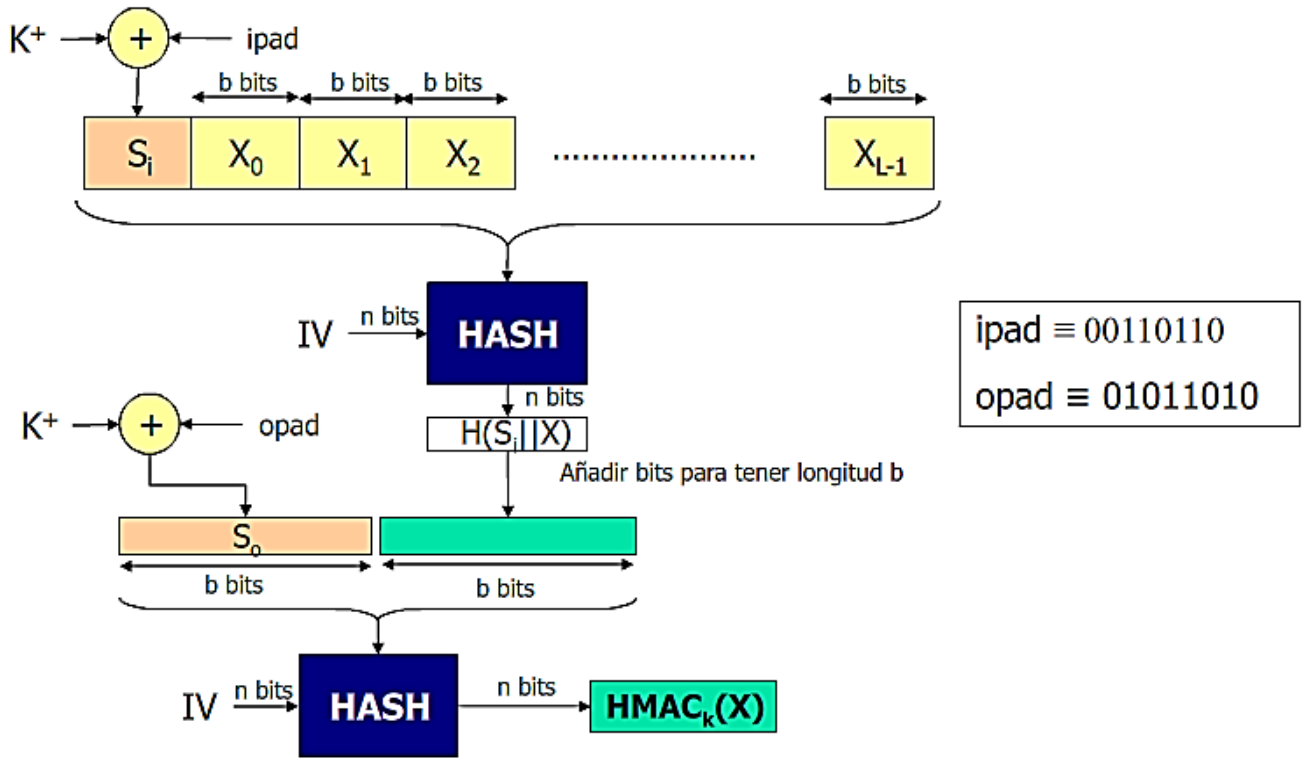


Figura 4. Pasos para la obtención de HMAC [Ref.10]

En el algoritmo se añaden ceros a la izquierda de clave k para crear una cadena de bits de longitud b→k+. Posteriormente se realizará la operación (k+)xor(ipad) para producir el bloque Si de b bits que se concatenará con el mensaje M.

Para finalizar se aplica la función de hash H al flujo generado y se realizará la operación (k+)xor(opad) para producir el bloque So de b bits que se concatenará con el código hash generado y se aplicará la función H al flujo generado dando como resultado final el HMAC.

2.3 ATAQUES SOBRE FUNCIONES HASH

Las funciones de dispersión pueden recibir ataques sobre dos de sus propiedades fundamentales que son la no presencia de colisiones y la unidireccionalidad.

Dependiendo del tipo de algoritmo criptográfico que se tenga existen numerosos tipos de ataques. Estos ataques pueden ser genéricos si se pueden aplicar a todas las funciones hash sin importar lo buenas que sean, o específicos si buscan los defectos de algún diseño en particular.

El principal tipo de ataque que se puede realizar es por fuerza bruta. En este tipo de ataque si la función no es unidireccional se seleccionan entradas aleatorias para el algoritmo y de entre todas ellas se busca una que proporcione el valor que buscamos, y en el caso de que la función presente colisiones lo que se buscará es un par de entradas que generen la misma salida.

Decimos que una función hash es de seguridad ideal si los mejores ataques que se conocen contra ella son de carácter genérico. Los criptoanalistas consideran que una función puede es vulnerable si su seguridad es menor que ideal lo cual no significaría que dicha función no pudiera ser utiliza en algunas aplicaciones.

2.3.1 ATAQUES POR FUERZA BRUTA. Ataque de cumpleaños

Dentro de los ataques por fuerza bruta existentes el más importante y conocido es el ataque del cumpleaños (birthday attack).

Se trata de una derivación de la paradoja del cumpleaños en la que se dice que la probabilidad de que dos o más personas de un grupo de 23 personas cumplan años el mismo día es superior al 50%.

Si pretendemos sacar una probabilidad mejor que el 50%, habría que evaluar la función sobre $1{,}2*2^{n/2}$ entradas elegidas al azar.

Este algoritmo se utiliza para reducir la complejidad algorítmica cuando se pretende encontrar dos mensajes con el mismo resumen.

Para hacerlo se sacan dos mensajes de dos conjuntos disjuntos y se comparan sus valores. En caso de que los resúmenes sean distintos sacamos un nuevo mensaje y lo compramos con los anteriores y así progresivamente hasta que se encuentre una colisión.

2.3.2 OTROS TIPOS DE ATAQUES

2.3.2.1 Ataque Wang-Yin-Yu o Ataque chino

Ataque creado por un grupo de matemáticos chinos que consiguieron romper la resistencia de los algoritmos SHA-1 y MD5.

Este tipo de ataque requiere menos esfuerzo que los ataques por fuerza bruta siendo que para realizar un ataque por fuerza bruta y encontrar dos mensajes M y M' aleatorios que tengan el mismo hash se necesitan generar por ejemplo, para el caso de SHA-1 de 160 bits, $2^{(160/2)}$ mensajes. Con este tipo de ataque en concreto se consigue encontrar para SHA1 una colisión del tipo de las "colisiones de cumpleaños" con un esfuerzo equivalente a 2^{69} operaciones hash en lugar de las 2^{80} operaciones hash requeridas por fuerza bruta. *[Ref.32]*

Como podemos ver Ataque chino es un tipo de ataque contra colisiones fuertes pero no contra las colisiones de 2 preimagen, a las cuales no afecta. Si alguien quiere generar un mensaje X' que tenga el mismo resumen SHA-1 que un mensaje X ya existente tiene que generar 2^{160} mensajes para conseguirlo.

2.3.2.2 Ataque Multicolisión de Joux

Antoine Joux demostró que la seguridad de las funciones hash en cascada es menor de $n2^{n/2}$ si una de las dos funciones que la componen es iterativa. Esta demostración quedó plasmada en la publicación en 2004 del estudio "Multicolisiones en funciones hash iterativas. Aplicación a construcciones en cascada".

Una colisión múltiple o multicolisión es una colección de varios mensajes hash con el mismo valor. El ideal de seguridad de una función de dispersión con una salida de n bits contra m maneras de ataques de colisión es proporcional a $2^{n(m-1)/m}$. Este argumento demuestra que la resistencia a las multicolisiones de una función hash iterativa es mucho menor que el ideal de seguridad por lo que Joux menciona que un ataque eficiente viene dado por una buen conocimiento de la construcción.

En la siguiente figura se muestra el ataque de Joux:

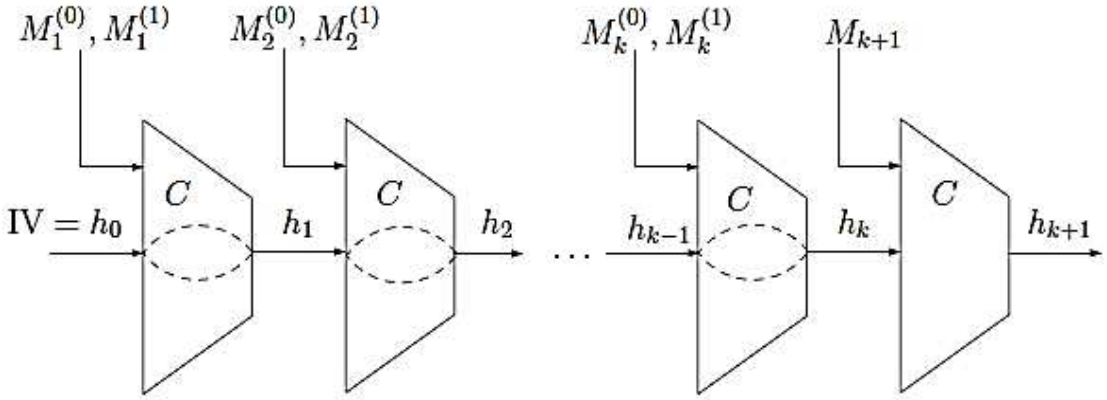


Figura 5. Ataque de Joux [Ref.07]

A continuación se estudiarán otros de los ataques contra los sistemas genéricos que utilizan funciones hash iterativas y que tiene como base la construcción Merkle-Damgard haciendo hincapié en el peligro de usar la funciones como cajas negras y revisando los ataques específicos sobre estas funciones hash estándares.

El concepto de caja negra viene dado por los ingenieros de software y los investigadores que tienden a reducir la complejidad de los sistemas pensando en los componentes de los sistemas como en cajas negras, es decir, módulos con una interfaz bien definida y con la funcionalidad y los detalles de implementación por separados. A su vez es común tratar a los objetos de programación como modelos de la vida real de abstracciones matemáticas.

La colisiones se suelen dar de manera teórica y no se llevan a la práctica dado que ningún protocolo natural las aceptaría pero con todo y eso se espera que las funciones hash sean resistentes a las colisiones. Es por esto que las colisiones no suelen suponer un peligro para los sistemas reales, pero esto no es siempre así como por ejemplo en el siguiente ataque en el que se demuestra que los ataques por colisión sobre la función de compresión de una función hash iterativa se pueden convertir en colisiones de mensajes con un perfecto significado.

Este ataque es el “ataque del bloque envenenado” en el cual se supone que existe un algoritmo que dado x de longitud de n bits encuentra dos cadenas de m bits M y M’ tal que C(x,M) = C(x,M’). Este ataque afecta sobre la construcción de Merkle-Damgard, en concreto sobre el bloque de nivel de la construcción y fue demostrado por Magnus Daum y Stefan Lucks en 2005 sobre el algoritmo MD5.

Ataque de preimagen: es el ataque que se produce cuando se intenta encontrar un valor arbitrario cuyo hash colisiones con otro hash. Es mucho más grabe que se dé el ataque de preimagen que el de colisión.

Las pseudo-colisiones son otro problema de estos algoritmos, las cuales son colisiones producidas en la función de compresión empleada en el proceso iterativo de una función de dispersión. Que haya pseudo-colisiones no implica que el algoritmo no sea seguro.

En la siguiente tabla se resumen algunos ataques específicos sobre las funciones estándar:

FUNCIÓN HASH	ATAQUES			
	Autor	**Tipo**	**Complejidad**	**Año**
MD4	Dobbertin	Colisión	2^{22}	1996
	Wang et. Al	Colision	2^{8}	2005
MD5	Dan Boer y Bosselaers	Pseudo-colisión	2^{16}	1993
	Dobbertin	Comienzo libre	2^{34}	1996
	Wang et. Al	Colisión	2^{39}	2005
SHA-0	Chabaud y Joux	Colisión	2^{61}	1998
	Biham y Chen	Casi-colisión	2^{40}	2004
	Biham et. Al	Colisión	2^{51}	2005
	Wang et. Al	Colisión	2^{39}	2005
SHA-1	Biham et. Al	Colisión(40 rondas)	Muy lento	2005
	Biham et. Al	Colisión(58 rondas)	2^{75}	2005
	Wang et. Al	Colisión(58 rondas)	2^{33}	2005
	Wang et. Al	Colisión	2^{63}	2005

Tabla 3. Ataques sobre Funciones Estándar [Ref.07]

2.3.4 INTENCIÓN DE LOS ATAQUES

Con los ataque lo que se quiere conseguir es:

TIPO DE HASH	METAS DEL DISEÑO	FUERZA IDEAL	METAS DEL TAQUE
Funciones Hash en un sentido (OWHF)	Resistencia Preimagen Resistencia 2ª Preimagen	2^n 2^n	Producir Preimágenes Producir 2ª Preimagen
Funciones Hash resistentes colisiones	Resistencia a colisiones	$2^{n/2}$	Producir colisiones

MAC	No recuperación de clave de longitud m y resistencia del mensaje	Min(2^m,2^n)	Encontrar la clave o producir un nuevo mensaje

Tabla 4. Intención de los ataques

2.4 APLICACIONES DE LAS FUNCIONES HASH EN CRIPTOGRAFÍA

Las aplicaciones de funciones hash en criptografía son numerosas. Algunos ejemplos son:

- En la construcción de utilidades más complejas como son las estructuras de datos y los algoritmos de cifrado y descifrado, cifradores de flujo y cifradores de bloque.
- En la construcción de herramientas utilizadas para proteger la integridad como son la firma digital o la suma de verificación y la prueba de integridad de contenidos.
- En la construcción de herramientas vinculadas a la autenticación y al control de acceso para autenticar entidades protegiendo y derivando claves.
- En la construcción de herramientas para la identificación y la rápida comparación de datos como huellas digitales.

Muchas de estas aplicaciones, como podemos ver, son relativas al campo de la criptografía puesto que ésta es la rama de las matemáticas encargada de proporcionar herramientas mediante las que conseguir seguridad en los sistemas informáticos.

CAPÍTULO 3
ALGORITMOS CRIPTOGRÁFICOS DE FUNCIÓN RESUMEN SHA Y MD5

3.1 ¿QUÉ SON LOS ALGORITMOS DE RESUMEN? SHA y MD5

Las funciones hash como MD5 y SHA-1 son algoritmos de resumen diseñados desde el principio con el objetivo de su utilización en sistemas criptográficos con requisitos de seguridad específicos. Hoy en día se han convertido en estándares para muchos desarrolladores y diseñadores de protocolos que las utilizan a pesar de no conocer ni sus propiedades ni su estructura ni su diseño.

La utilización de algoritmos de resumen sin un conocimiento pleno de ellos no fue cuestionada hasta el año 2004 en el que se encontraron debilidades sobre ellos mediante el incremento del número de ataques y la calidad de los mismos.

En la siguiente tabla se muestran algunas de las características de los algoritmos de resumen SHA, MD y Whirlpool:

NOMBRE	TAMAÑO BLOQUE (bits)	TAMAÑO PALABRA (bits)	TAMAÑO SALIDA (bits)	RONDAS	AÑO DEL ESTANDAR
MD4	512	32	128	48	1990
MD5	512	32	128	64	1992
SHA-0	512	32	160	80	1993
SHA-1	512	32	160	80	1995
SHA-256	512	32	256	64	2002
SHA-384	1024	64	384	80	2002
SHA-512	1024	64	512	80	2002
SHA-224	512	32	224	64	2004
Whirlpool	512	-	512	10	2003

Tabla 5. Características SHA, MD y Whirlpool [Ref.07]

3.1.1 CONSTRUCCION MERKLE DAMGARD

La estructura Merkle-Damgard es una construcción recursiva llamada así en honor a R. Merkle y a I. Damgard pioneros en utilizarla de forma independiente en 1989.

Su estructura es la siguiente:

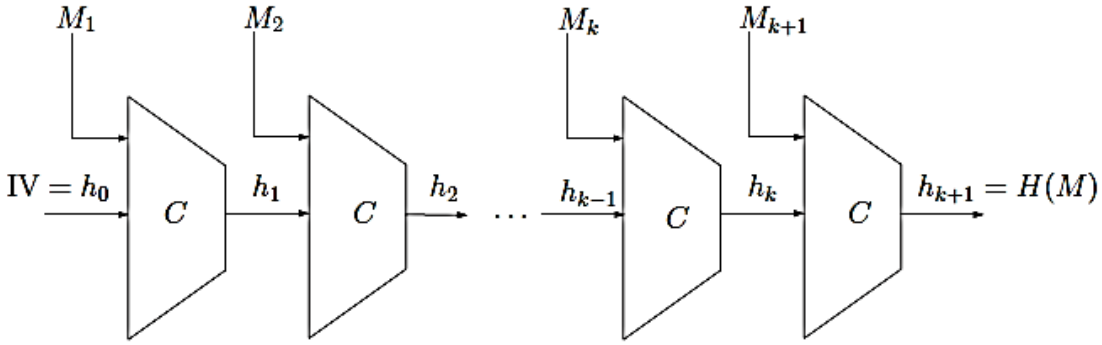


Figura 5. Estructura Merkle-Damgard [Ref.07]

- El número de bits del mensaje de entrada M es de longitud arbitraria y múltiplo de la longitud de los bloques. Para conseguir que el mensaje M sea múltiplo del tamaño de los bloques se debe alargar la longitud de M mediante bits relleno hasta conseguir una longitud aceptable.
- El mensaje de entrada M se divide en bloques de longitud fija obteniendo así un conjunto de bloques M1,..., Mk.
- Se realiza un proceso iterativo en el cual:
    - h_0=IV: se toma un vector inicial.
    - $h_i=f(x_i,h_{i-1})$, i=1,2,...,k: se operara mediante una función de compresión f y se utiliza el resultado de cada operación como parte de la entrada en siguiente paso.
    - $H(M)=g(H_k)$: el resultado final es el producto de operar todos los subbloques de mensaje M_1,..., M_k con los resultados obtenidos en cada paso.

En este tipo de construcciones hay dos elecciones importantes que afectarán a las propiedades de la función que son la elección de la regla de relleno o padding y la elección de valor o vector inicial (IV) que vendrá definido como parte de la descripción de la función hash.

Para poder comprender mejor esta construcción debemos explicar de qué están compuestas las construcciones seguras.

Se puede decir que la construcción segura constan de 2 componentes que pueden ser estudiados por separado, que son la función de compresión la cual proporcionará una entrada de longitud fija a una salida de longitud fija también, y la extensión del dominio, que producirá una función con una entrada de longitud arbitraria mediante una función de compresión.

Las funciones hash de un solo sentido o OWHF no pueden construir funciones hash resistentes a colisiones. Las funciones hash resistentes a colisiones están basada en cifradores por bloques.

La prueba de seguridad de estas y otras construcciones basadas en el cifrado por bloques asumen que los cifrados subyacentes son indistinguibles de cierta abstracción, llamado el ideal de cifrado, el cual va más allá de los requisitos de seguridad estándares para estos sistemas.

En cuanto al segundo componente, la extensión del dominio, podemos decir que la más sencilla y más utilizada es la construcción Merkle-Damgard, la cual funciona de la siguiente manera:

Dada una función de compresión C con las siguientes características:

- $C: \{0,1\}^n \times \{0,1\}^m \mapsto \{0,1\}^n$.
- Un valor inicial (IV) de n bits.
- Y dada a la entrada un mensaje M con las siguientes características:
- Partiendo el mensaje M en bloques $M_1,\ldots,M_k$, rellenando si es necesario.
- Teniendo M_{K+1} cifrado |M|.
- Teniendo h_0=IV.

- Para i=1 a k+1 tenemos $h_i=C(h_{i-1},M_i)$.
- Salida h_{k+1}.

En la construcción se itera la función de compresión C con el bloque de mensaje M correspondiente. La salida de cada iteración se convierte en la entrada de la siguiente aplicación de C y así sucesivamente hasta que en la última iteración se obtiene el hash del mensaje completo H(M). Los valores temporales de las salidas de las funciones de compresión hi se llaman variables de encadenamiento o estados internos.

En los dos primeros pasos de esta construcción hay cierta flexibilidad y cualquier codificación se realizará siempre y cuando cumplan tres condiciones fundamentales que son que M se codifica como un número entero de m bits, que la codificación ha de estar siempre libre de colisiones y que la longitud de M vendrá dada en el último bloque.

Algo importante es que la actualización de alguna parte del mensaje por pequeña que sea puede desencadenar el recalculo de todo el hash.

Si la función de compresión es resistente a las colisiones entonces la construcción resultante también lo será, pero este tipo de construcción, es decir la construcción Merkle-Damgard, produce una función con muchas más propiedades estructurales y con una serie de vulnerabilidades inesperadas.

Esta construcción es la razón fundamental de que sea erróneo pensar que las funciones hash son cajas negras.

La construcción iterativa fue diseñada con el propósito de extender el dominio de las funciones resistentes a las colisiones por lo que no se debe esperar que ofrezcan garantías de seguridad más lejos de su extensión.

3.1.2 WHIRLPOOL

Whirlpool es una algoritmo diseñado por Paulo Barreto y Vicente Rijmen que fue presentado como respuesta a la convocatoria de primitivas criptográficas emitida por el NESSIE (Nuevos esquemas europeos de firma, integridad y cifrado).

Whirlpool fue seleccionado junto con SHA-256, SHA-384, SHA-512, como parte de la elección del NESSIE.

Este diseño combina la extensión del dominio Merkle-Damgard con un cifrador por bloques basado en una función de compresión que es una variante del AES y por consecuente contra-opuesto a SHACAL, y se convirtió la función de compresión utilizada en la construcción Miyaguchi-Preneel.

No se especifica la arquitectura aunque el procesado de 32 bits o 64 bits permiten algunas optimizaciones posibles que no se podían hacer con implementaciones de 8 bits.

3.2 MD5

El algoritmo MD5 junto con SHA-1 fueron los dos algoritmos más utilizados entre la segunda mitad de los años 90 y la primera mitad de la década siguiente.

El algoritmo MD5 es un algoritmo para la creación de resúmenes basado en los principios de diseño de las funciones de compresión que utilizan cifradores de bloques y en una extensión del dominio que viene dada por la construcción recursiva Merkle-Damgard.

Las letras "MD" hacen referencia a "Message Digest" y los números a las distintas versiones del algoritmo.

Su antecesor, MD4, fue propuesto por Ron Rivest en 1990 y poco después, concretamente en 1992, fue remplazado por MD5, siendo éste una versión más fuerte y mejorada respecto a seguridad. El diseño de ambos algoritmos tendrá una gran influencia en sucesivas construcciones de funciones hash, y en especial, como veremos posteriormente, en el diseño de SHA-1.

Las características principales del algoritmo MD5 son las siguientes:

- Su función de compresión opera con bloques de 512 bits subdivididos en 16 bloques de 32 bits.
- El tamaño de su estado interno, es decir, de la variable cadena, y de su salida son de 128 bits. Ésta última está representada como un número de 32 dígitos en hexadecimal.
- El número de rondas, es decir, el número de cambios secuenciales de su situación interna, es de 64 y están organizadas en una red de Feistel.
- Cada ronda utiliza un subbloque de mensaje de 32 bits para actualizar el estado interno a través de una mezcla no lineal de operaciones booleanas y aritméticas.

- Cada subbloque de 32 bits es utilizado cuatro veces por la función de compresión.
- Se asignan 64 bits de relleno al último bloque de mensaje para que su longitud sea congruente con 448 módulo 512.

Las similitudes entre los algoritmos MD5 y MD4 son que ambos tienen necesidad de relleno y el mismo tamaño de resumen. Además ambos constan de cuatro rotaciones aunque entre ellas sí que existen ligeras diferencias.

Las diferencias más notables entre ambos algoritmos son:

- MD5 tiene una cuarta vuelta.
- En MD5 cada paso tiene una sola constante aditiva.
- En MD5 la función G en la vuelta 2 ha pasado de ser G(X,Y,Z)=(XY)v(XZ)v(YZ) a ser G(X,Y,Z)=(X∧Z)v(Y∧¬Z) para conseguir que la función G sea menos simétrica.
- En MD5 cada paso se añade al resultado del paso anterior.
- En el nuevo algoritmo, el orden en el que se accede a las palabras de entrada en las rondas 2 y 3 se ha cambiado consiguiendo que estos patrones sean más diferentes entre sí.
- En MD5 el número de desplazamientos en cada vuelta ha sido optimizado.
- MD5 es un poco más lento que MD4.

En la siguiente figura se muestran los pasos a seguir hasta la obtención del resumen final:

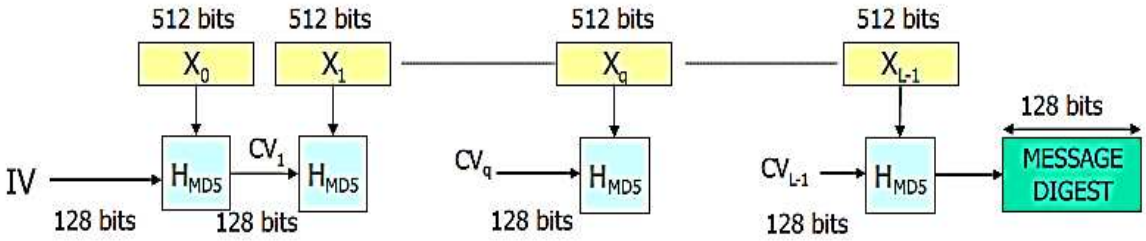


Figura 6. Pasos en MD5 para la obtención del resumen [Ref.10]

El algoritmo MD5 consta de 5 pasos:

- Adición de bits: se añaden bits hasta que la longitud sea congruente 448 módulo 512. Esta operación se llama padding o relleno.
- Paso de longitud del mensaje: se concatena al resultado anterior un entero de 64 bits y para conseguir un múltiplo exacto de 512. Esta operación se denota attaching.
- Inicialización del búfer con cuatro palabras: en este paso se preparan los buffers internos con unos valores predeterminados que son siempre los mismos.

Palabra A: 01 23 45 67

Palabra B: 89 ab cd ef

Palabra C: fe dc ba 98

Palabra D: 76 54 32 10

- Procesado del mensaje: se procesa el mensaje en bloques de 16 palabras mediante funciones auxiliares. A la entrada se tendrán palabras de 32 bits y se producirá como salida palabras de 32 bits. A pesar de ser la parte más opaca del algoritmo, es más importante. Las funciones que se utilizan son las siguientes:

$$F(X,Y,Z)=(X \wedge Y) \vee (\neg X \wedge Z)$$

$$G(X,Y,Z)=(X \wedge Z) \vee (Y \wedge \neg Z)$$

$$H(X,Y,Z)=X \oplus Y \oplus Z$$

$$I(X,Y,Z)=Y \oplus (X \vee \neg Z)$$

Las operaciones utilizadas son XOR (⊕), AND (v), OR (∧) y NOT (¬) y las funciones utilizadas son F, G, H e I las cuales actúan bit a bit en paralelo.

- Paso de salida: el resumen del mensaje será la salida producida por A, B, C y D, comenzando por el byte de menor peso de A y acabando por el byte de mayor peso de D.

La estructura del resumen MD5 es la siguiente:

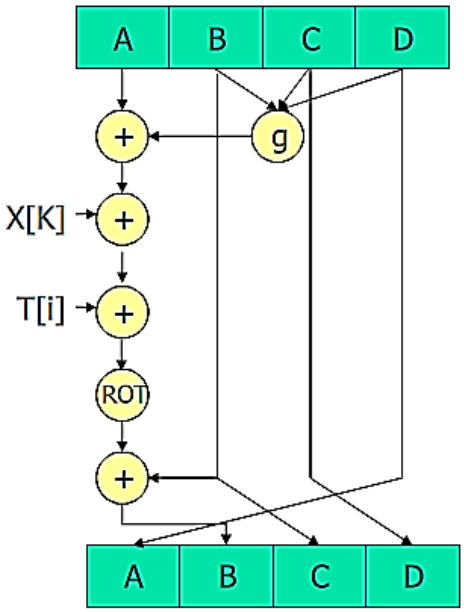


Figura 7. Esquema 1 Resumen MD5 [Ref.10]

Para procesar los bloques hay cuatro rondas con una función lógica diferente, que son, F, G, H e I. En el siguiente gráfico se muestra la progresión por las distintas funciones:

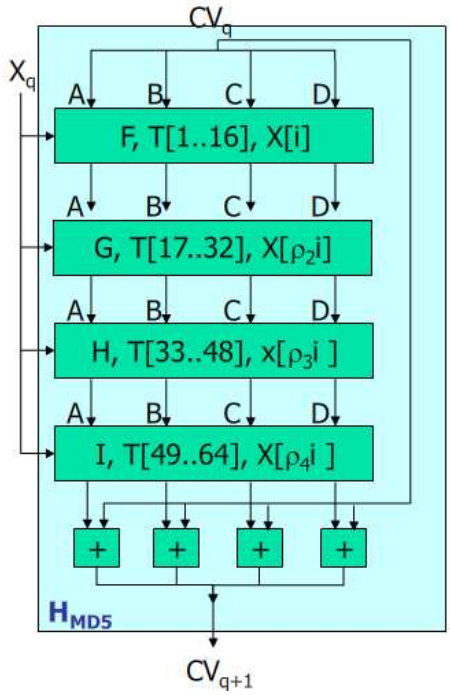


Figura 8. Esquema 2 Resumen MD5 [Ref.10]

Donde H_{MD5} es donde se comprime el mensaje y $X_0 \ldots X_{L-1}$ son los bloques del mensaje.

Es un algoritmo sencillo y que funciona bien en arquitecturas de 32 bits. En cuanto a seguridad se refiere, la probabilidad de encontrar dos mensajes que generen el mismo resumen es del orden de 2^{64} operaciones y la probabilidad de encontrar un mensaje conociendo el resumen es del orden de 2^{128} operaciones.

Este algoritmo se suele utilizar para comprobar que archivos descargados de Internet no hayan sido modificados. Esto nos protege de troyanos y de virus y además nos indica si la descarga realizada está corrupta o incompleta. En los sistemas UNIX se utiliza para calcular el hash de las claves de los usuarios y así poder, mediante la utilización de claves públicas y privadas, determinar si un correo electrónico ha sido alterado o no.

Actualmente se han demostrado sus debilidades y se recomienda su sustitución por algoritmos como SHA-1 o RIPEMD-160. Se ha determinado que es un algoritmo muy vulnerable frente ataques de fuerza bruta como podría ser el ataque del cumpleaños.

El algoritmo fue criptoanalizado con técnicas similares a las que se utilizaron para criptoanalizar al algoritmo MD4 y como resultado se encontraron pseudocolisiones en la función de compresión aunque no en el algoritmo completo. Adicionalmente, se estimó que era posible construir una máquina capaz de atacar el algoritmo por fuerza bruta y encontrar una colisión en 24 días siendo, en 1994, el coste de dicha máquina de 10 millones, lo cual resulto inviable.

3.3 SHA

Las siglas SHA hacen referencia a Secure Hash Algorithm o lo que es lo mismo Algoritmo de Hash Seguro. Existen tres versiones del algoritmo: SHA-0 publicado en 1993 y retirado poco después, SHA-1 y SHA-2.

La estructura de los 3 algoritmos es distinta pero existen numerosas similitudes entre SHA-0 y SHA-1. Para la realización del diseño del algoritmo SHA-1 simplemente se corrigieron algunas deficiencias que tenía de base el algoritmo SHA-0. Es por esto por lo que SHA-0 no fue utilizado en muchas aplicaciones y fue rápidamente sustituido por SHA-1.

La estructura del algoritmo SHA-2 si sufrió un cambio significativo respecto a la estructura del algoritmo SHA-1. SHA-2 nunca llegó a sustituir al algoritmo SHA-1 a pesar de que en éste se corregían deficiencias existentes en el diseño de SHA-1.

En la siguiente tabla se muestra una comparativa de los algoritmos SHA-0, SHA-1 y de las versiones del algoritmo SHA-2:

Algoritmo		**Tamaño de la salida (bits)**	**Tamaño del estado interno (bits)**	**Tamaño del bloque (bits)**	**Tamaño máximo del mensaje (bits)**	**Tamaño de la palabra (bits)**	**Rondas**	**Operaciones que se realizan**	**Existencia de colisiones**
SHA-0		160	160	512	2^{64}	32	80	xor, and y or	Sí
SHA-1									Ataque Teórico (2^{51})
SHA2	SHA-256/224	256/224	256	512	2^{64}	32	64	xor, and y or	No
	SHA-512/384	512/384	512	1024	2^{128}	64	80		

Tabla 6. Comparativa SHA-0, SHA-1 y SHA-2 [Ref.75]

3.3.1 SHA-1

SHA-1 es un algoritmo criptográfico implementado por la NSA, Agencia de Seguridad Nacional, en 1995 y publicado por el NIST, Instituto Nacional de Estándares y Tecnología, como sucesor de su antecesor SHA-0.

Su estructura es similar a la del algoritmo MD4 pero con algunos matices del algoritmo MD5.

Este algoritmo convierte archivos de casi cualquier longitud en una línea fija de unos y ceros sobre la que se realizan numerosas operaciones y numerosas mezclas durante múltiples ciclos para producir como resultado un resumen final llamado hash.

En la siguiente figura se muestran los pasos que son necesarios seguir para la obtención del resumen final:

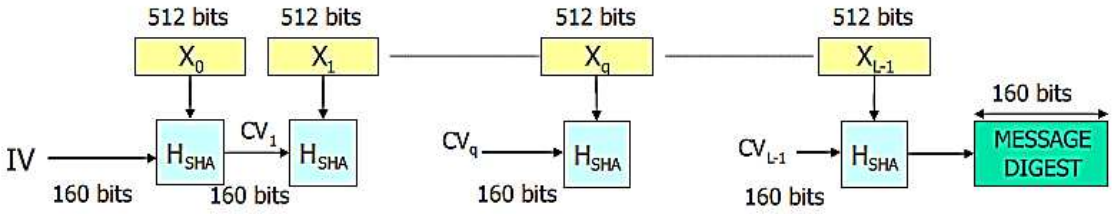


Figura 9. Pasos en SHA-1 para la obtención del resumen [Ref.10]

Algunas de las características principales del algoritmo son:

- Para poder tratar con bloques de 512 bits y conseguir una longitud máxima de mensaje de 264 bits se utiliza un sistema de relleno.
- El tamaño de su estado interno y la longitud de su salida son de 160 bits.

- Posee 80 funciones de ronda variadas y elaboradas.
- Su complejidad algorítmica es de 2^{80} operaciones.
- Su vector inicial es un registro de 160 bits inicializado por los siguientes valores:

  Palabra A= 67 45 23 01
  Palabra B= EF CD AB 89
  Palabra C= 98 BA DC FE
  Palabra D= 10 32 54 76
  Palabra E= C3 D2 E1 F0

  Los pasos a seguir para la obtención del resumen final son los siguientes:

- Adición de bits de relleno hasta conseguir una longitud final de 448 módulo 512.
- División del mensaje en bloques de 512 bits.
- Inicialización del buffer con los valores iniciales mencionados anteriormente.
- Operaciones y desplazamientos.

  El esquema en SHA-1 para la obtención del resumen es el siguiente:

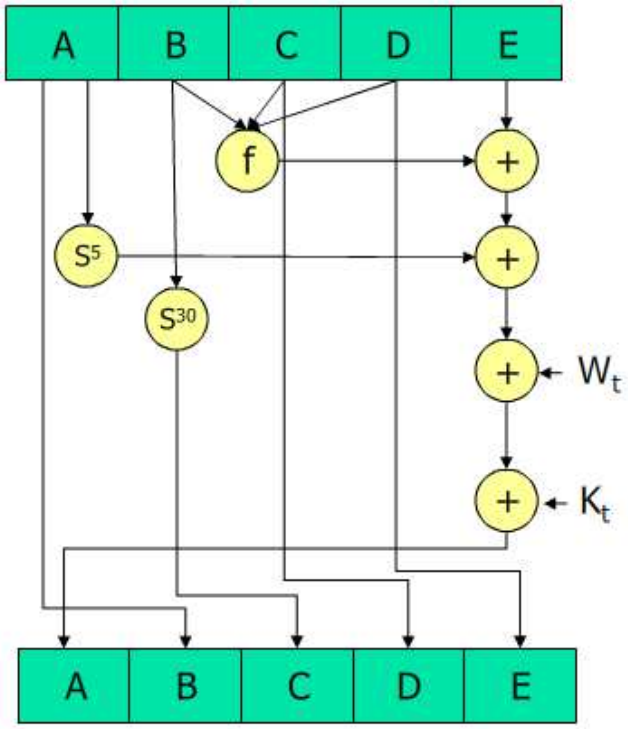


Figura 10. Esquema 1 Resumen SHA-1[Ref.10]

Sobre A se ejecutará la función (E + f(t,B,C,D) + S^5(A)+W + K), donde:

- A, B, C, D, y E son las palabras de 32 bits del buffer. El procedimiento para la obtención de los subbloques de 32 bits, de los 512 bits iniciales de mensaje, es complejo y como consecuencia si un bit del mensaje origen cambia, más de la mitad de los subbloques cambiaran.
- t es el número de operación.
- S(A) es el desplazamiento circular de bits hacia la izquierda. Este desplazamiento puede ser de 5 o 30 bits (en el ejemplo es de 5).
- W es la palabra de 32 bits derivada de la entrada de 512 bits.
- W=S1 (Wt-16 ⊕ Wt-14 ⊕ Wt-8 ⊕ Wt-3)
- Después de la última operación se produce un desplazamiento del registro hacia la derecha.
- K es una constante aditiva.

En la siguiente imagen se muestra el esquema del resumen en el que los bloques de 512 bits se operan a través de los módulos H_{SHA}:

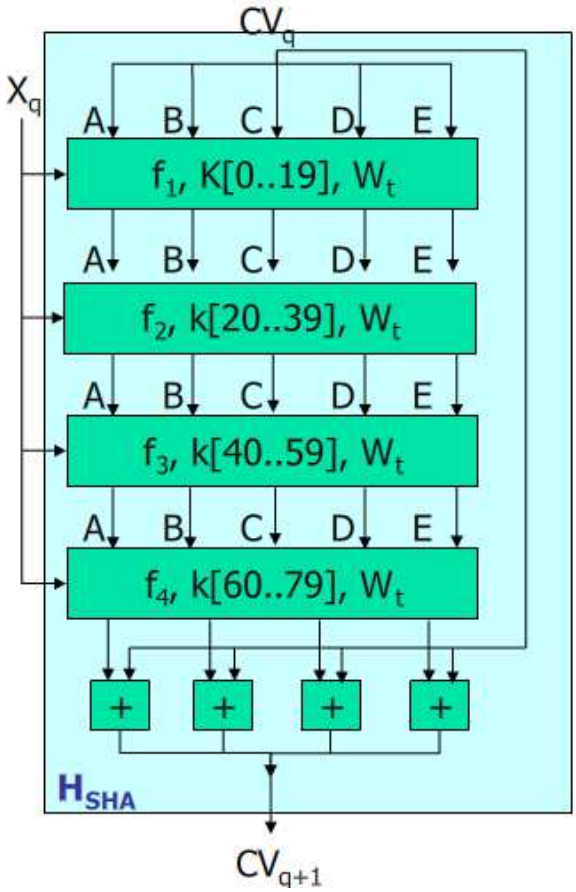


Figura 11. Esquema 2 Resumen SHA-1[Ref.10]

Cada módulo H_{SHA} pasa por 4 rondas con 20 operaciones cada una de ellas. En cada ronda se opera mediante cuatro tipos de funciones diferentes f1 f2 f3 y f4, una para cada ronda, tomando una constante diferente k comprendida entre 0 y 80.

Las cuatro funciones utilizadas son:

$$f1 = f(t,B,C,D) = (B \wedge C) \vee (B \wedge D)$$

$$f2 = f(t,B,C,D) = B \oplus C \oplus D$$

$$f3 = f(t,B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$$

$$f4 = f(t,B,C,D) = B \oplus C \oplus D$$

Las cuatro constantes k utilizadas son:

k= 5A 82 79 99

k= 6E D9 EB A1

k= 8F1B BC DC

k= CA 62 C1 D6

Es un algoritmo sencillo que funciona bien en arquitecturas de 32 bits. Respecto a su seguridad, la probabilidad de encontrar dos mensajes que generen el mismo resumen es del orden de 2^{80} operaciones y la probabilidad de encontrar un mensaje conocido el resumen es del orden de 2^{160}.

Es el algoritmo existente más utilizado y está presente en múltiples aplicaciones de seguridad como TLS, SSL, IPSec. Otro tipo de aplicación en la que se utiliza es en videoconsolas, en concreto en la Nintendo Wii, en la que es utilizado para la verificación de la firma durante el arranque. Pero sin duda, la utilidad más importante de este algoritmo, es la de estándar de certificación digital, autenticando de manera univoca personas o entidades.

Aunque al sistema de cifrado que opera dentro de la función de compresión nunca se le ha dado ningún reconocimiento oficial, esto no ha detenido a la comunidad de criptoanalistas a aislarlo y estudiarlo, hasta que finalmente en 2005 se encontraron deficiencias en su seguridad. Esto supuso un gran problema al afectar la caída de SHA-1 a la seguridad de los certificados digitales pudiendo generar estos con la misma firma que otros, lo que suponía la posible suplantación de identidad de personas o de páginas Web.

Actualmente el algoritmo está debilitado en un 99% en relación con su fortaleza inicial y la recomendación que se está dando es su sustitución por el algoritmo RIPEMD-160 hasta la futura implantación del nuevo algoritmo SHA-3.

3.3.2 SHA-2

El algoritmo SHA-2 es un conjunto de funciones criptográficas diseñadas por la Agencia de Seguridad Nacional (NSA) entre los años 2002 y 2004. El conjunto de sus versiones lo forman los algoritmos SHA-224, SHA-256, SHA-384 y SHA-512 desarrollados en 2002 y el algoritmo SHA-224 desarrollado en 2004.

Los algoritmo SHA-256 y SHA-512 tienen un diseño similar respecto a lo acometido por el NIST. En SHA-256 el tamaño de la salida es de 256 bits, el tamaño del estado interno es de 256 bits, el tamaño del bloque es de 512 bits, el tamaño máximo del mensaje es de 2^{64} bits y se opera con palabras de 32 bits, y en SHA-512 el tamaño de la salida es de 512 bits, el tamaño del estado interno es de 512 bits, el tamaño del bloque es de 1024 bits, el tamaño máximo del mensaje es de 2^{128} bits y se opera con palabras de 64 bits. Ambos diseños tienen gran parecido con SHA-1, aunque están mucho más cerca entre sí que de su antecesor común. SHA-384 dista de SHA-512, en una modificación trivial consistente en el recorte de la salida a 384 bits y en el cambio del valor inicial de la variable de la cadena. Por otro lado, SHA-224 se define como una versión truncada de SHA-256 con un valor inicial diferente y un tamaño de salida 224 bits.

La diferencia más importante entre los cuatro nuevos algoritmos y SHA-1 es el procedimiento para la obtención de los subbloques del bloque del mensaje.

Las operaciones que se realizan son las siguientes:

$$Ch(E,F,G)=(E \wedge F) \oplus (\neg E \wedge G)$$

$$Ma(A,B,C)=(A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\sum 0(A)=(A>>>2) \oplus (A>>>13) \oplus (A>>>22)$$

$$\sum 1(E)=(E>>>6) \oplus (E>>>11) \oplus (E>>>25)$$

En la siguiente figura se muestra el esquema para la obtención del resumen:

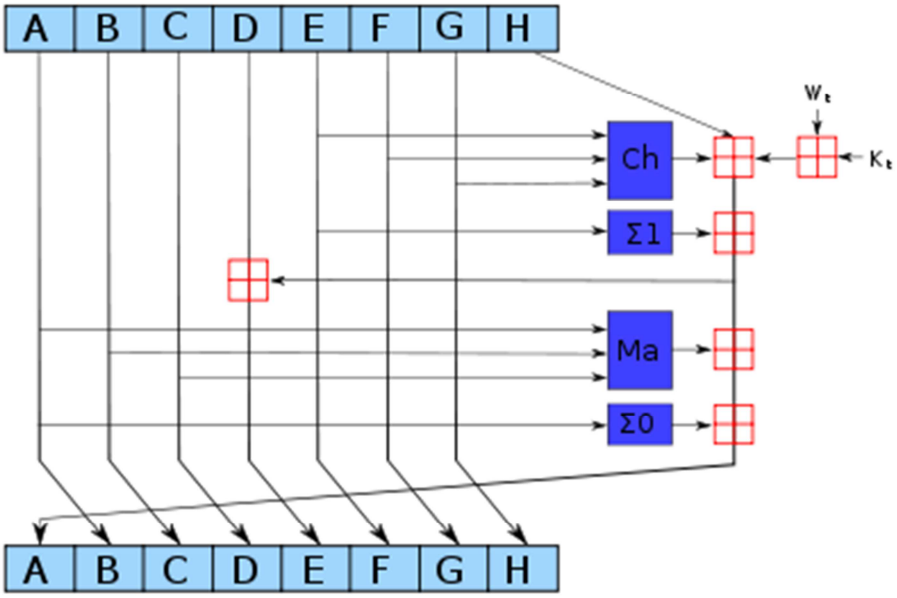


Figura 12. Figura algoritmo SHA-2

Cuando en 2005 se encontraron deficiencias en la seguridad de SHA-1 se comprobó que SHA-2 al presentar múltiples diferencias con el algoritmo anterior, aunque algunas similitudes, los ataques no se extendieran con éxito.

3.4 DIFERENCIAS ENTRE SHA-1 Y MD5

MD5 y Sha-1 tienen similitudes y diferencias y a continuación se enumerar las más significativas:

SIMILITUDES:

- Sha-1 fue desarrollado de una forma similar a MD4, teniendo algunas connotaciones de MD5. Ambos utilizan el mismo algoritmo de relleno, dividiendo el mensaje en bloques de 512 bits y codificando la longitud como un número 64 bits. Sha-1 genera una salida de 160 bits de longitud mientras que MD5 genera una salida de sólo 128 bits por lo que la dificultad de generar un mensaje que tenga un resumen dado para MD5 es de 2^{128} operaciones y de 2^{160} operaciones para Sha-1. Además la dificultad de generar dos mensajes aleatorios distintos y que tengan el mismo resumen para MD5 es de 2^{64} operaciones y para SHA-1 es de 2^{80} operaciones *[Ref.10]*. Esta diferencia de 16 bits a favor de Sha-1 lo convierte en más seguro y resistente a ataques por fuerza bruta que el algoritmo MD5, aunque es más lento que éste.
- Ambos algoritmos procesan bloques de 512 bits y emplean 4 funciones primitivas para generar el resumen del mensaje, pero SHA-1 realiza un mayor número de pasos que MD5, es decir, Sha-1 realiza 80 frente a MD5 que realiza 64. SHA1 tiene procesar 160 bits de buffer en comparación con los 128 bits de MD5, por lo que la ejecución del algoritmo SHA-1 es más lenta que la de MD5. Dichas las funciones de ronda en MD5 son más simples y menos variadas que en SHA-1.
- SHA-1 utiliza un procedimiento más complejo para obtener los subbloques de 32 bits de los 512 bits iniciales de mensaje. Si un bit del mensaje cambia, más de la mitad de los subbloques cambian, lo cual no pasaba en MD5 en el que solo variaban 4.

DIFERENCIAS

- La longitud máxima del mensaje para SHA-1 debe ser menor de 264 bits mientras que para MD5 no hay limitaciones de longitud.

- El algoritmo MD5 emplea 64 constantes (una por cada paso) mientras que el algoritmo Sha-1 sólo emplea 4 (una para cada 20 pasos). Por último cabe mencionar que MD5 se basa en la arquitectura little-endian mientras que Sha-1 se basa en la arquitectura big-endian.

CAPÍTULO 4

SHA-3

SHA-3 es un concurso que impulso el NIST entre los años 2007 y 2012 con el objetivo de encontrar "el algoritmo más seguro del mundo" *[Ref.02]*.

El NIST desde un principio intentó encontrar un algoritmo de hash que convirtiera un mensaje de longitud variable en un breve "resumen del mensaje" y que se pudiera utilizar de estándar para la generación de firmas digitales, la autenticación de códigos de mensaje, y para muchas otras aplicaciones de seguridad en la infraestructura de la información.

El concurso comenzó hace varios años y se lograron 5 finalistas. La competición culminó con la elección del ganador en 2012, y su resultado fue llamado "SHA-3" que será inscrito en el FIPS, Federal Information Processing Standard, o lo que es lo mismo, Estándares Federales de Procesamiento de la Información, y que se utilizará como futuro estándar a nivel mundial.

Todo empezó en 2004, cuando la criptoanalista china, Xiaoyun Wang, descubrió debilidades en el algoritmo "Secure Hash Algorithm-1″ (SHA-1), que era el algoritmo de resumen utilizado en multitud de aplicaciones importantes, como en la firma digital, en transacciones bancarias, o en el almacenamiento de claves, entre otras muchas cosas.

En febrero de 2005 se anunció la vulnerabilidad demostrada en el SHA-1, y se realizó una reunión con el objetivo de evaluar su estado. Algunos de los participantes

recomendaron la transición del algoritmo SHA-1 al algoritmo SHA-2, pero al final se decidió que lo más prudente era realizar un nuevo algoritmo a través de concurso público.

Los requisitos mínimos de seguridad que determinó el NIST y que todas las funciones que se presentaron al concurso debían que cumplir eran:

- Al menos una variante de la función hash debe ser compatible con HMAC y con hash asignados al azar.
- Para cada valor de cada bit del resumen, la función hash debe proporcionar resistencia contra las preimágenes.
- Resistencia a segundas preimágenes.
- Resistencia a las colisiones.
- Debe ser resistente a ataques de grandes longitudes.
- Se debe poder proporcionar resúmenes de mensaje de 224, 256, 384 y 512 bits para permitir la sustitución de la familia SHA-2.

El periodo de tiempo que se ha estipulado para la realización de este concurso es de 6 años. En un principio se presentaron las bases y los participantes inscribieron sus algoritmos. En esta primera inscripción se presentaron 64 propuestas, entre las cuales estaban las presentadas por equipos tan ilustres como IBM, France Telecom, etc., de las que tras muchos análisis, en 2010 se redujeron a 14 para posteriormente reducir a 12 candidatos cuando se encontraron ataques de preimagen para dos de ellos. Durante todo este periodo de tiempo, el NIST ha dado plazos para poder realizar pequeñas modificaciones en los algoritmos que han ido siendo seleccionados, y hoy por hoy, de estos 14 algoritmos solo quedan 5 que son los finalistas. La elección de estos cinco finalistas fue tomada, según el NIST, atendiendo a la fortaleza y el diseño de los algoritmos.

Estos 5 finalistas son: BLAKE, algoritmo creado por la compañía Nagravisión en Suiza, SKEIN, creado por el estadounidense Bruce Schneier y sus compañeros,

KECCAK, diseñado por un equipo Belga, GROESTL, diseñado en colaboración de las universidades de Graz en Austria y la Universidad Técnica de Dinamarca y JH, creado por Hongjun Wu.

El plazo para realizar los ajustes oportunos terminó el 16 de enero de 2010, y durante ese año cualquier criptógrafo pudo analizar dichos algoritmos, hasta que a finales de 2012 se proclamó el ganador.

La idea de la competición fue inspirada por lo que hizo el gobierno de Estados Unidos con su estándar AES (Advanced Encryption Standard), el cual fue elegido de forma parecida, es decir, en forma de concurso.

4.1 ANTECEDENTES, fase previa (2004 / 2007)

Entre los años 2004 y 2005, cuando la comunidad criptoanalista se percató de las deficiencias, en cuanto a seguridad se refería, que poseía el estándar gubernamental de resumen utilizando hasta el momento, SHA-1, el NIST ante la alarma social que se estaba produciendo decidió, mediante la creación de dos talleres, analizar los algoritmos que habían sido aprobados por la organización hasta la fecha solicitando para ello la ayuda de la comunidad criptoanalista y pidiendo su opinión sobre la política gubernamental que se había estado siguiendo hasta el momento en cuanto a la elección de algoritmos de resumen estándares.

Como resultado de esta búsqueda de opiniones y de estos análisis de algoritmos la organización decidió sustituir el algoritmo de resumen que se había estado utilizando hasta la fecha promoviendo para ello un concurso público. En este concurso se elegiría el nuevo algoritmo de resumen SHA-3 que sería utilizado de estándar mediante la selección de algoritmos de resumen que hubieran sido presentados al concurso y que hubieran pasado con éxito por sucesivas fases eliminatorias que debería tener. En 2006 se publicó un cronograma orientativo sobre cómo se desarrollaría el concurso y se publicaron las bases de utilización de funciones hash y el concurso comenzó a cobrar vida.

En Enero de 2007 el NIST anunció para su discusión los requisitos mínimos que debían cumplir los algoritmos que fueran a ser presentados al concurso, así como los criterios que se tendrían en cuenta a la hora de la selección en la primera fase eliminatoria. La organización revisó las bases que habían propuesto basándose en la respuesta pública obtenida y publicó el 2 de Noviembre de 2007 un segundo y final informe que pondría en marcha el concurso.

4.2 EL CONCURSO (2007 / 2012)

El 2 de Noviembre de 2007 el NIST anunció en una publicación del Registro Federal, el equivalente a nuestro BOE (Boletín Oficial del Estado), la competición pública que se llevaría a cabo y que tenía como objetivo el encontrar el nuevo algoritmo de hash criptográfico que se utilizaría como estándar y que tendría como nombre SHA-3.

Para el 31 de Octubre de 2008 se habían recibido 64 propuestas de criptoanalistas de todo el mundo de las cuales, en Diciembre de 2008, se seleccionaron 51 como candidatas de la primera ronda. Posteriormente en Julio de 2009, de entre estas 51 propuestas, se seleccionaron 14 como candidatas a la segunda ronda y finalmente en Diciembre de 2010 de entre estas últimas 14 se eligieron las 5 finalistas: BLAKE, Grøstl, JH, Keccak y Skein. Estos 5 algoritmos pasarían a formar el grupo de algoritmos participantes en la tercera ronda y de entre ellos se elegiría el algoritmo ganador.

Desde el primer momento la comunidad criptoanalista fue participe del concurso a través de sus comentarios enviados directamente al NIST, de opiniones publicadas en un foro público y de cripto-análisis y de análisis de rendimiento de las funciones candidatas publicados en las principales publicaciones criptográficas o expuestos en conferencias sobre criptografía. Por su parte el NIST organizó una conferencia pública por cada una de las ronda del concurso con el objetivo de recoger información. Finalmente el 2 de Octubre de 2012, basándose en estos comentarios y en la revisión interna de los candidatos, se anunció como ganador del concurso SHA-3 al algoritmo Keccak dando así por finalizado el concurso que había durado 5 años.

4.3 LAS FASES

4.3.1 PRIMERA RONDA (10 de Diciembre de 2008 / 24 de Julio de 2009)

Al principio del concurso se habían presentado un total de 64 candidaturas pero de entre todas ellas solo se pudieron seleccionar de 51 algoritmos por ser los que cumplían con el mínimo de requisitos establecidos por el NIST. El 9 de Diciembre de 2008 se publicó esta selección de 51 candidatos como participantes en la primera ronda y el 10 de Diciembre de 2008 comenzó el concurso.

Entre los días 25 y 28 de Febrero de 2009 el NIST organizó la primera conferencia con los candidatos a SHA-3 en la universidad católica de Lovaina, Bélgica. Los objetivos de esta conferencia eran la presentación por parte de los participantes de sus algoritmos y la discusión del proceso a seguir y de los criterios que se tendrían en cuenta en la selección de los candidatos de la segunda ronda.

El 24 de Julio de 2009 el NIST anunció los 14 finalistas de esta ronda y éstos pasarían a ser los algoritmos candidatos de la segunda ronda.

4.3.2 SEGUNDA RONDA (28 de Septiembre de 2009 / 9 de Diciembre de 2010)

El 28 de Septiembre de 2009 el NIST publicó los 14 participantes en la segunda. Estos 14 candidatos habían sido seleccionados de entre los 51 candidatos de la primera ronda. La segunda ronda comenzó.

Se destinó un año a revisión pública de los algoritmos y tras ello el NIST organizó la conferencia correspondiente a esta fase del concurso SHA-3. La conferencia tuvo lugar en la universidad de California, Santa Bárbara, entre los días 23 y el 24 de

Agosto de 2010 y que tuvo como objetivo discutir la seguridad y el rendimiento de los candidatos de esta ronda.

El 9 de Diciembre de 2010 y tras la buena acogida que había tenido la iniciativa del análisis y del estudio de los algoritmos por parte de la comunidad criptoanalista, el NIST basándose en estos estudios y en un análisis interno de los algoritmos anunció los 5 finalistas que conformarían el grupo de participantes de la tercera y final ronda, dando así por terminada la segunda ronda de la competición.

4.3.3 TERCERA RONDA (31 de Enero de 2011 / 2 de Octubre de 2012)

El 31 de Enero de 2011 se publicaron los 5 finalistas elegidos en la segunda ronda, BLAKE, Grøstl, JH, Keccak, y Skein, y la tercera ronda comenzó.

Se destinó un año a revisión pública de estos algoritmos tras el que el NIST organizó la tercera conferencia, que tuvo lugar en Washington D.C. entre los días 22 y 23 de Marzo de 2012, que tendría como objetivo volver a discutir la seguridad y el rendimiento de estos algoritmos finalistas.

La comunidad criptoanalista volvió a acoger de muy buen agrado la iniciativa de la revisión publica de los algoritmos y el NIST finalmente, volviéndose a basar en estos análisis y en otra revisión interna de los algoritmos que llevó a cabo, el 2 de Octubre de 2012 publicó el ganador de la competición otorgando la victoria al algoritmo Keccak y dando así por finalizada competición que había tenido una duración total de 5 años.

4.4 LOS 5 FINALISTAS

Los 14 algoritmos candidatos en la segunda ronda eran:

- BLAKE de Jean-Philippe Aumasson
- BMW (Blue Midnight Wish) de Svein Johan Knapskog
- CubeHash de Dan Bernstein
- ECHO de Henri Gilbert
- Fugue de Charanjit S. Jutla
- Groestl de Lars R. Knudsen
- Hamsi de Özgül Küçük
- JH de Hongjun Wu
- Keccak de The Keccak Team
- Luffa de Dai Watanabe
- Zablah (Shabal) de Jean-François Misarsky
- SHA Vite-3 de Orr Dunkelman
- SIMD de Gaëtan Leurent
- Skein de Bruce Schneier

Al finalizar la segunda ronda se seleccionaron 5 de ellos como finalistas y candidatos de la tercera ronda.

4.4.1 CONSIDERACIONES PREVIAS

Las características y datos que se muestran a continuación han sido obtenidos de los informes presentados por los participantes para la segunda ronda. Este hecho es importante puesto que durante todas las fases del concurso se dio tiempo a los

participantes para modificar aspectos de sus algoritmos por lo que las especificaciones de las funciones podrían haber variado en los informes presentados en la tercera fase.

Algunos de los gráficos que se muestran a continuación están extraídos de un documento en el que se analiza el rendimiento sobre hardware utilizando FPGAs.

4.4.1.1 TERMINOLOGIA UTILIZADA EN LOS GRAFICOS

Para el correcto entendimiento de los gráficos explicativos de las funciones finalistas se muestra a continuación terminología utilizada en ellos:

X[i]	Referencia a una posición i en un array X
Xi	Referencia al bit de posición i en X
b	Tamaño de bloque en bits
h	Tamaño del valor del hash en bits
w	Tamaño de la palabra en bits
IV	Vector de inicialización
SEXT	Extensión del registro
ZEXT	Extensión cero
<<<R	Rotación hacia la izquierda R posiciones: Si R es una constante, se utiliza rotación fija Si R es una variable, se utiliza mecanismo de rotación
>>>R	Rotación a la derecha R posiciones: Si R es una constante, se utiliza rotación fija Si R es una variable, se utiliza mecanismo de rotación
<<S	Desplazamiento a la izquierda S posiciones: Si S es una constante, se utiliza desplazamiento fijo Si S es una variable, se utiliza mecanismo de desplazamiento

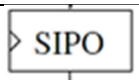
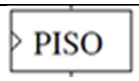
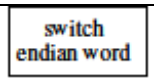
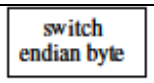
>>*S*	Desplazamiento a la derecha S posiciones: Si Si S es una constante, se utiliza desplazamiento fijo Si S es una variable, se utiliza mecanismo de desplazamiento
||	Concatenación. Por defecto los buses vuelven a tener la misma disposición que antes de la división
	Serial en la unidad paralela hacia afuera
	Paralelo en la unidad serial hacia afuera
	Cambio de orden en las palabras
	Cambio de orden en los bits

Tabla 7. Terminología utilizada

4.1.1.2 CONSTRUCCIONES UTILIZADAS

En la siguiente tabla se muestra una relación de los algoritmos BLAKE, Grøstl, JH, Keccak y Skein con algunas de las construcciones que utilizan:

ALGORITMO	CONSTRUCCIÓN UTILIZADA
BLAKE	HAIFA
GROESTL	SPONGE/MERKLE-DAMGARD
JK	SPONGE
KECCAK	SPONGE
SKEIN	MERKLE-DAMGARD/ TREEFISH

Tabla 8. Construcciones utilizadas en los algoritmos finalistas

MERKLE-DAMGARD

Construcción recursiva explicada con anterioridad en el apartado SHA-1 y MD5.

HAIFA (Hash Iterative Framework)

El diseño Haifa fue desarrollado por Eli Biham y Orr Dunkelman con el objetivo de mejorar la construcción Merkle-Damgard cuando sobre esta se encontraron colisiones.

La solución que se propone mediante esta construcción es manteniendo el diseño de Merkle-Damgard pero incrementando el tamaño del hash y definiendo familias de funciones hash aumentar la seguridad y el número de plataformas en las que se puede utilizar.

La cualidad más importante que posee la construcción Haifa es que permite el cálculo del hash en una sola pasada y con una cantidad de memoria fija independientemente del tamaño del mensaje.

TREEFISH

El cifrador Treefish es un cifrador de bloque construido por Niels Feguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas y Jesse Walker, es decir, por el equipo de desarrollo del algoritmo Skein, con el objetivo de su utilización exclusivamente para el concurso SHA-3.

Este cifrador está definido por 3 tamaños de bloque diferentes de 256, 512 y 1024 bits y por una clave de cifrado de igual tamaño que el tamaño de bloque utilizado. Su valor de ajuste es de 128 bits para todos los tamaños de bloque.

Consta de 3 operaciones fundamentales: suma, XOR y rotaciones fijas, y se utilizan para todas ellas palabras de 64 bits. Las versiones de 256 y 512 bits operan en 72 rondas y la versión de 1024 bits opera en 80 rondas.

La velocidad de cifrado de datos es buena y es debido a la utilización de operaciones sencillas y a su diseño concebido exclusivamente para ello desde el principio. En concreto la velocidad para el modelo de 512 bits es de 6,1 ciclos de reloj por byte y de 6,5 ciclos de reloj por byte para el modelo de 1024 bits.

Inicialmente se pensó como una mejora del cifrador de bloque simétrico Twofish, cifrador diseñado también por Bruce Schneier y publicado en 1998.

Algunos aspectos importantes de destacar del cifrador Twofish es que tiene relación con el crifrador Blowfich siendo que ambos tienen un tamaño de bloque de 128 bits, utilizan claves de 128, 192 o 256 bits y poseen un diseño flexible. Por otro lado también cabe destacar que Twofish consta de 16 rondas y que su diseñado fue orientado desde un principio hacia el buen rendimiento que debía tener pudiendo ser utilizado por ejemplo en CPUs de 32 bits y en tarjetas inteligentes de 8 bits.

CONSTRUCCIÓN ESPONJA

La construcción esponja es una función basada en una permutación de longitud fija o transformación y en una regla de relleno o padding que permite la creación de una función de entrada y de una función de salida de longitud variables.

En esta construcción se toma como entrada una cadena binaria de longitud arbitraria y se devuelve como salida una cadena binaria de longitud arbitraria que será requerida por el usuario. Se opera en un estado finito y se aplican iterativamente permutaciones internas intercalando para ello la entrada de la entrada o la recuperación de la salida.

La permutación a la que se llama no debe tener ninguna propiedad estructural común a las permutaciones aleatorias normales. Su estado interno se compone de una matriz de 5x5 de palabras de 64 bits y la entrada es absorbida en el estado del hash a un ritmo determinado.

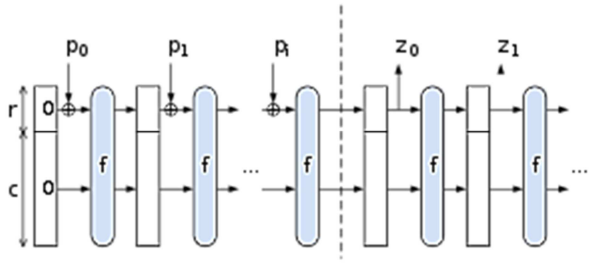


Figura 13. Construcción esponja [Ref.93]

CIFRADOR CHA-CHA

El cifrador ChaCha es una función de cifrado de flujo de 256 bits basada en el cifrador Salsa20 pero más resistente además de conseguir preservar la velocidad.

Estos cifradores están construidos por una función aleatoria de 32-bit y operaciones de adición y rotaciones de bits para producir una salida de 512 bits.

Su estado inicial se compone de 16 palabras, todas ellas dispuestas en una matriz de 4x4, que se operan durante 20 rondas para producir una salida también de 16 palabras.

El tamaño de su estado interno es de 16 palabras de 32 bits e internamente el sistema de cifrado utiliza 32 bit adicionales y una constante para las operaciones de rotación.

En cada ronda se realizan operaciones de cuatro cuartos de vuelta sobre la matriz de 4x4 y cada 2 rondas el patrón se repite. En cada cuarto de vuelta se modifican cuatro palabras.

4.4.2 BLAKE

El algoritmo Blake es una función hash criptográfica desarrollada por Jean-Philippe Aumasson, Luca Henzen, Willi Meier, y Raphael C.-W. Phan con el objetivo de presentarla como candidata en el concurso SHA-3.

Es una función muy sencilla de implementar que está basada en el cifrador de flujo Chacha de Dan Bernstein y que mantiene una estructura similar a la estructura HAIFA con pequeñas modificaciones como son la adición de palabras al mensaje y la inversión de las direcciones de rotación.

Al igual que el algoritmo SHA-2 se presenta en dos variantes, BLAKE-32 que utiliza palabras de 32 bits para conseguir resúmenes de hasta 256 bits de longitud y BLAKE-64 que utiliza palabras de 64 bits para conseguir resúmenes de hasta 512 bits de longitud. La transformación del bloque central se realiza combinando 16 palabras de entrada con 16 variables de trabajo y manteniendo solo 8 palabras entre los bloques.

Este algoritmo consta de 6 pasos: inicialización, función de ronda, paso en columna, paso en diagonal, función de ronda y finalización. A continuación se muestra mediante imágenes la transición entre los distintos pasos:

INICIALIZACIÓN

Los 16 estados se inicializan tal que diferentes entradas producen diferentes estados iniciales:

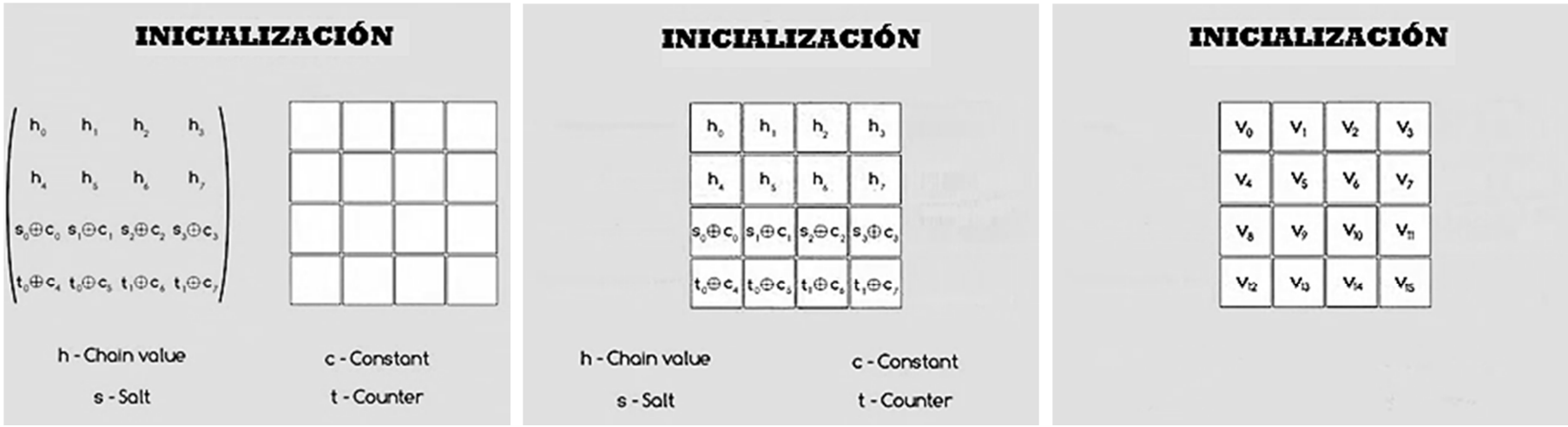


Figura 14. Inicialización algoritmo BLAKE [Ref.40]

FUNCIÓN DE RONDA

La función de ronda itera 14 veces en BLAKE-32 y 16 veces en BLAKE-64 sobre el estado v.

Cada ronda implica transformaciones sobre el estado v basadas en la función principal G.

Blake utiliza la función principal G y la repite 112 veces en BLAKE-32 y 128 veces en BLAKE-64.

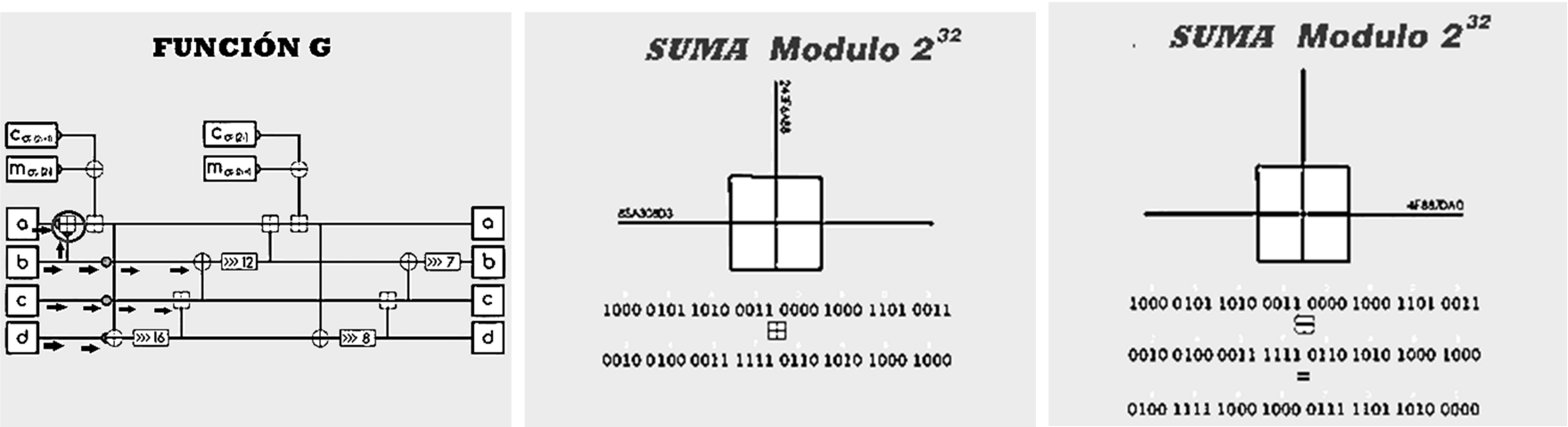


Figura 15. Función G suma algoritmo BLAKE [Ref.40]

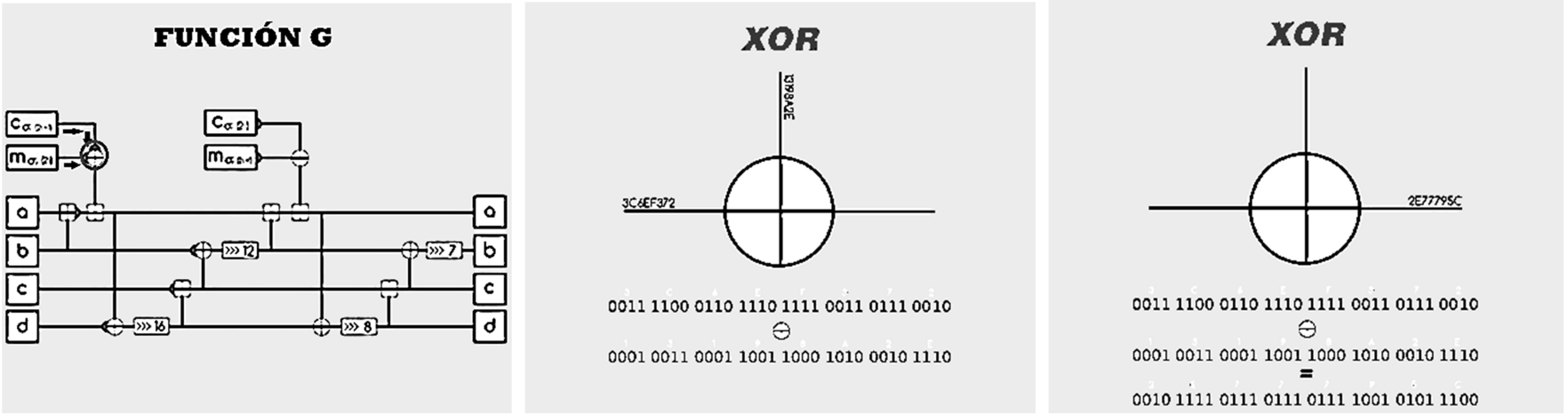


Figura 16. Función G XOR algoritmo BLAKE [Ref.40]

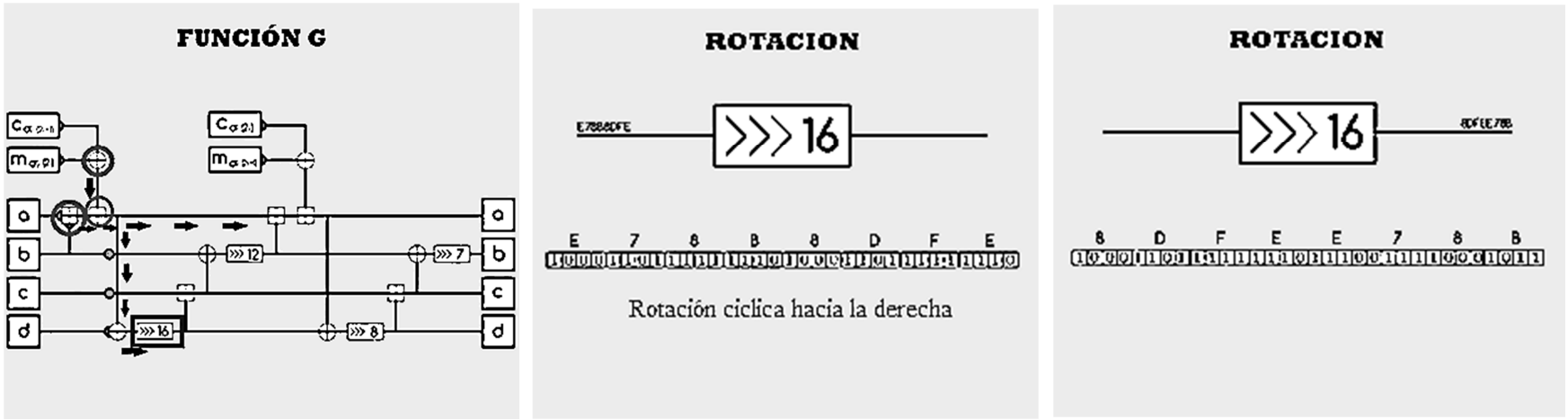


Figura 17. Función G rotación algoritmo BLAKE [Ref.40]

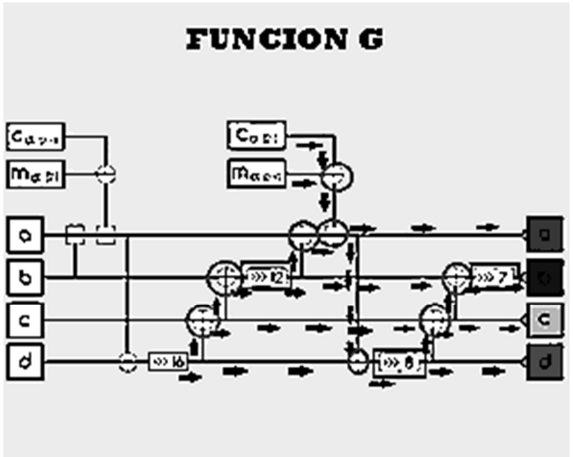


Figura 18. Función G salida algoritmo BLAKE [Ref.40]

3_Criteria_Hardware_revised.pdf

PASO EN COLUMNA

Se aplica G a cada columna. Las primeras cuatro llamadas son operadas en paralelo.

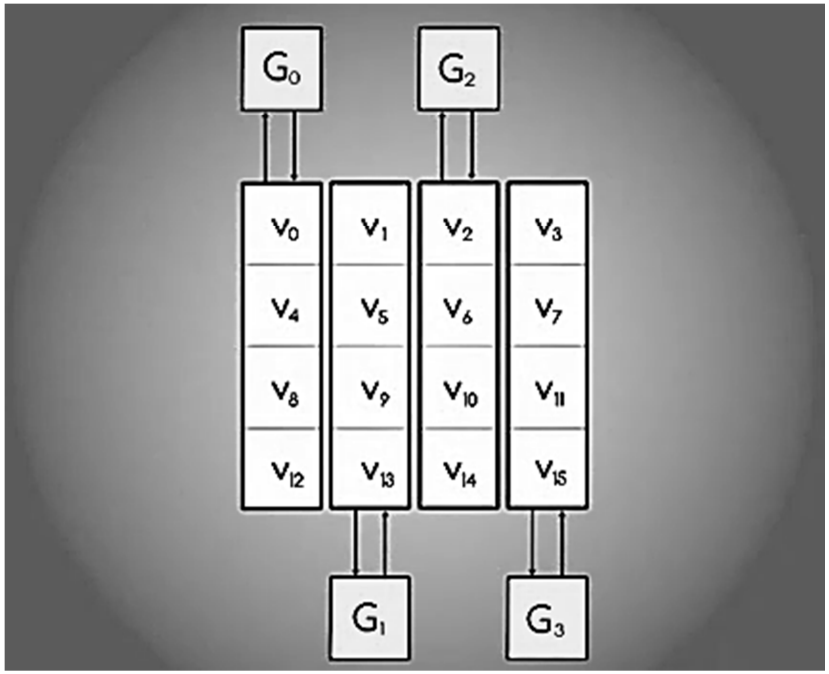


Figura 19. Paso en columna algoritmo BLAKE [Ref.40]

PASO EN DIAGONAL

Se aplica G a cada diagonal en paralelo.

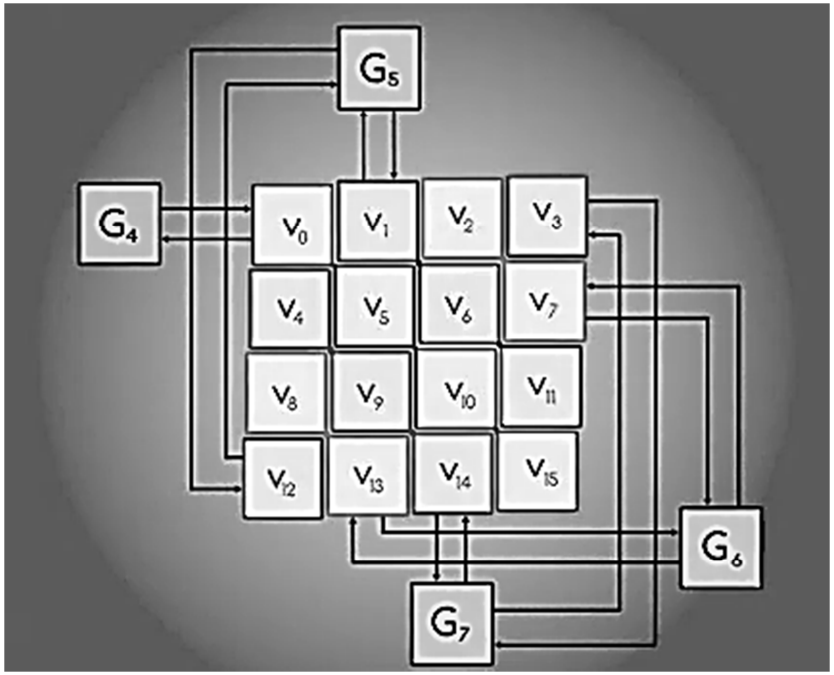


Figura 20. Paso en diagonal algoritmo BLAKE [Ref.40]

FUNCION DE RONDA

Se opera sucesivamente: primera ronda 8 llamadas, segunda ronda 16 llamadas, tercera ronda 24 llamadas y así sucesivamente se van acumulando las llamadas hasta llegar a la ronda 14 en la que se habrán realizado un total de 112 llamadas.

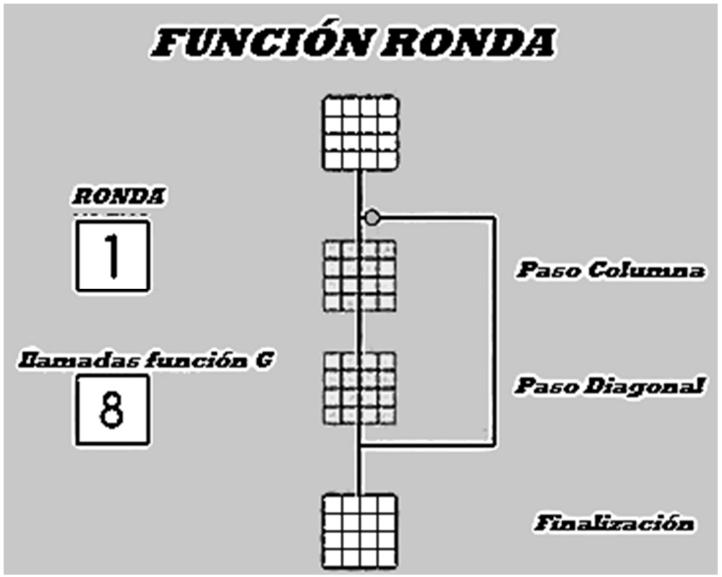


Figura 21. Inicio función ronda algoritmo BLAKE [Ref.40]

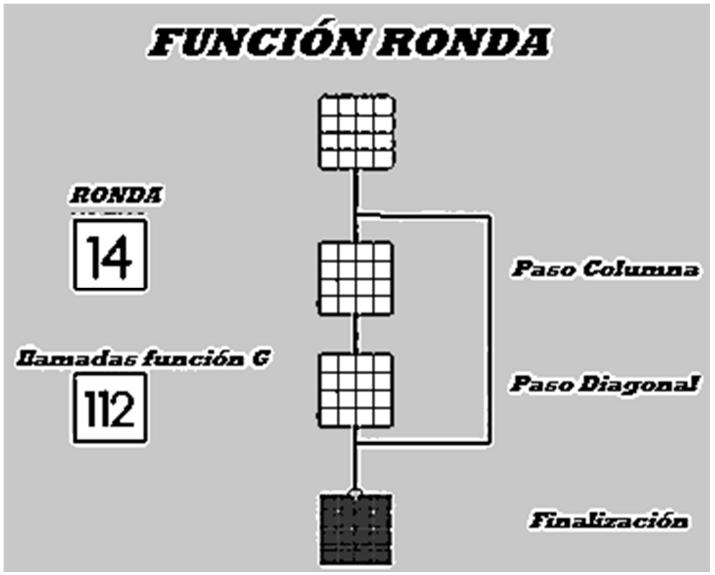


Figura 22. Final función ronda algoritmo BLAKE [Ref.40]

FINALIZACIÓN

Una nueva cadena de valores de h'_0-h'_{15} se extrae del estado v_0-v_{15}.

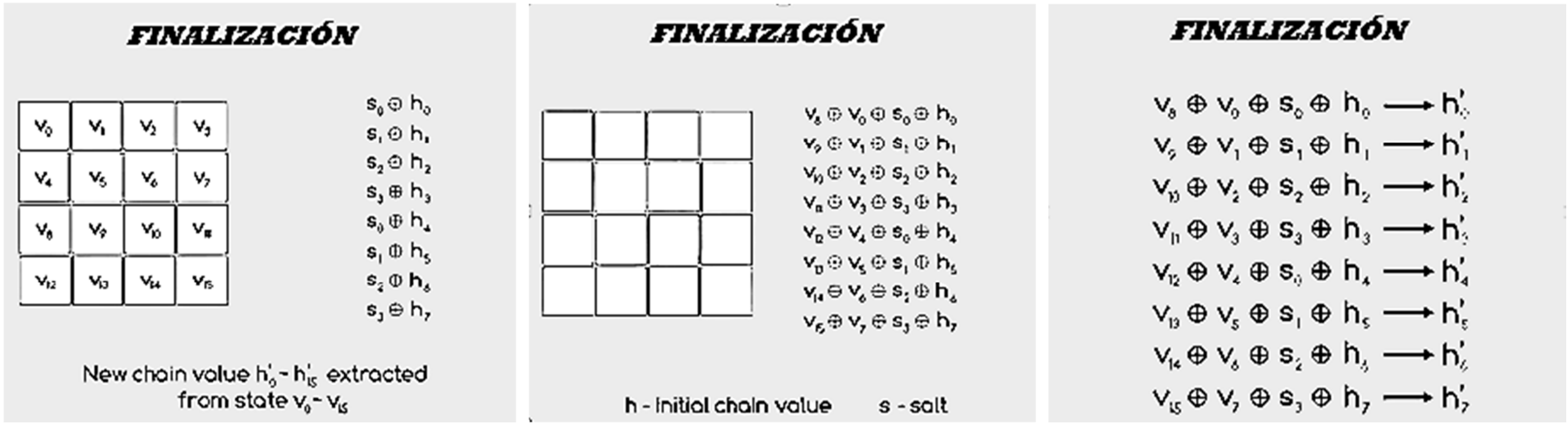


Figura 23. Finalización algoritmo BLAKE [Ref.40]

A continuación se mostraran gráficos sobre la versión del algoritmo BLAKE que fue presentada en la segunda ronda del concurso.

En la siguiente figura se muestra el diseño del algoritmo utilizado para analizar su rendimiento de hardware en FPGAs:

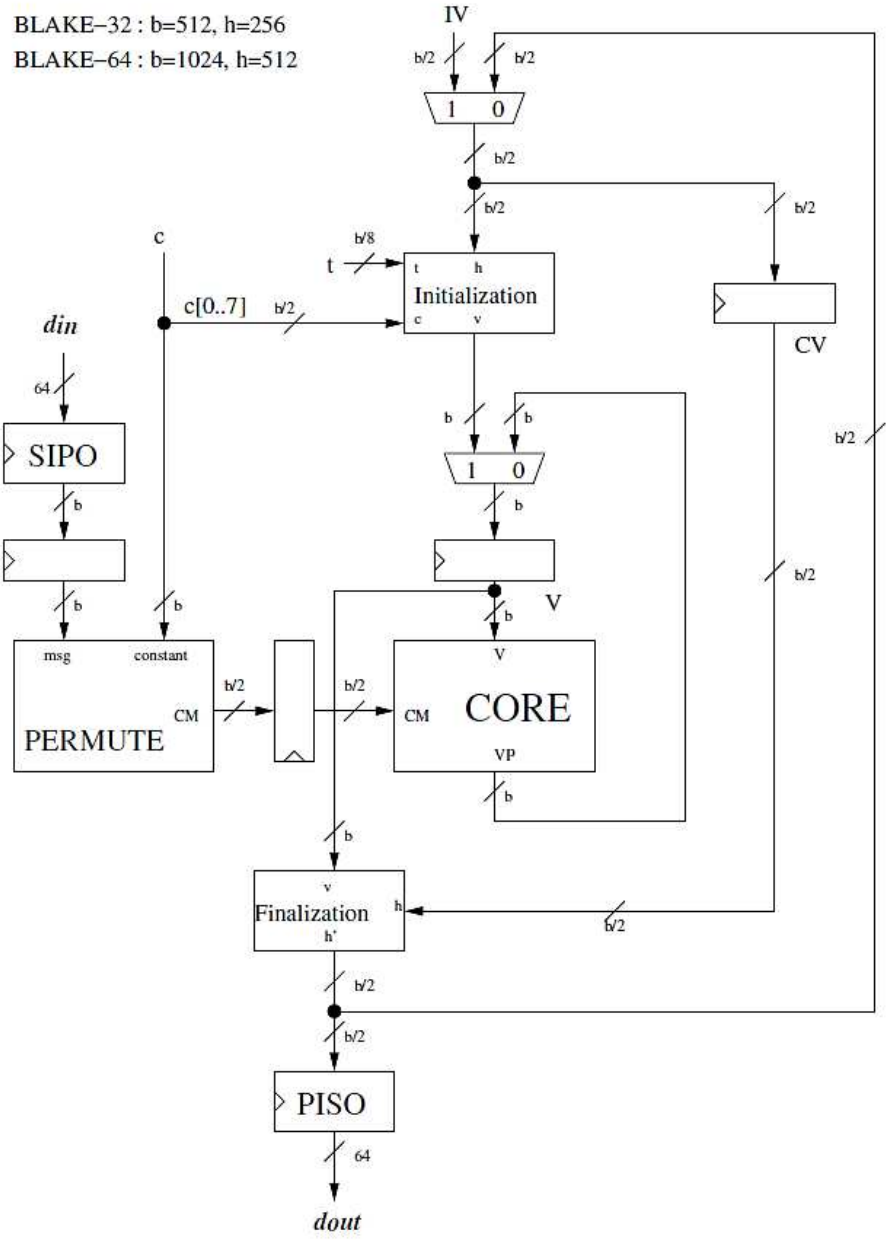


Figura 24. Diseño algoritmo BLAKE [Ref.64]

Para empezar un bloque de mensaje de entrada se carga en la unidad *SIPO* y se almacena en un registro temporal utilizado para poder mantenerlo hasta que éste pueda ser procesado por el *CORE*. Para cuando este bloque de mensaje pueda ser procesado por el *CORE*, se habrá cargado simultáneamente el siguiente bloque de mensaje en la *SIPO*. Después el bloque de mensaje *msg* y la constante c entran en la unidad de permutación de la función *PERMUTE* produciendo como salida el bloque permutado CM que posteriormente será pasado como uno de los datos de entrada del *CORE*.

Por otro lado y simultáneamente, el valor de la cadena *CV* se inicializa con un valor inicial *IV* y la vez que la entrada *V* del *CORE* es inicializa con un valor que dependerá del valor la cadena inicial, del contador t y de la mitad inferior de la constante *c*.

La cadena *V* y el bloque permutado *CM* son mezclados por el núcleo o *CORE* durante 10 rondas, consumiendo para ello dos ciclos de reloj para completar una ronda. La salida del *CORE* o *VP* se utiliza como valor de la siguiente cadena del siguiente bloque de mensaje o como valor final del hash en caso de que sea el último bloque de mensaje a procesar.

La matriz de operaciones que se sigue en la unidad de Inicialización es la siguiente:

$$\begin{pmatrix} v[0] & v[1] & v[2] & v[3] \\ v[4] & v[5] & v[6] & v[7] \\ v[8] & v[9] & v[10] & v[11] \\ v[12] & v[13] & v[14] & v[15] \end{pmatrix} \longleftarrow \begin{pmatrix} h[0] & h[1] & h[2] & h[3] \\ h[4] & h[5] & h[6] & h[7] \\ c[0] & c[1] & c[2] & c[3] \\ t[0] \oplus c[4] & t[1] \oplus c[5] & c[6] & c[7] \end{pmatrix}.$$

Las operaciones que se realizan en la unidad de Finalización son las siguientes:

$$
\begin{aligned}
h'[0] &\leftarrow h[0] \oplus v[0] \oplus v[8] \\
h'[1] &\leftarrow h[1] \oplus v[1] \oplus v[9] \\
h'[2] &\leftarrow h[2] \oplus v[2] \oplus v[10] \\
h'[3] &\leftarrow h[3] \oplus v[3] \oplus v[11] \\
h'[4] &\leftarrow h[4] \oplus v[4] \oplus v[12] \\
h'[5] &\leftarrow h[5] \oplus v[5] \oplus v[13] \\
h'[6] &\leftarrow h[6] \oplus v[6] \oplus v[14] \\
h'[7] &\leftarrow h[7] \oplus v[7] \oplus v[15]
\end{aligned}
$$

En la siguiente figura se muestran las operaciones que se realizan en una permutación:

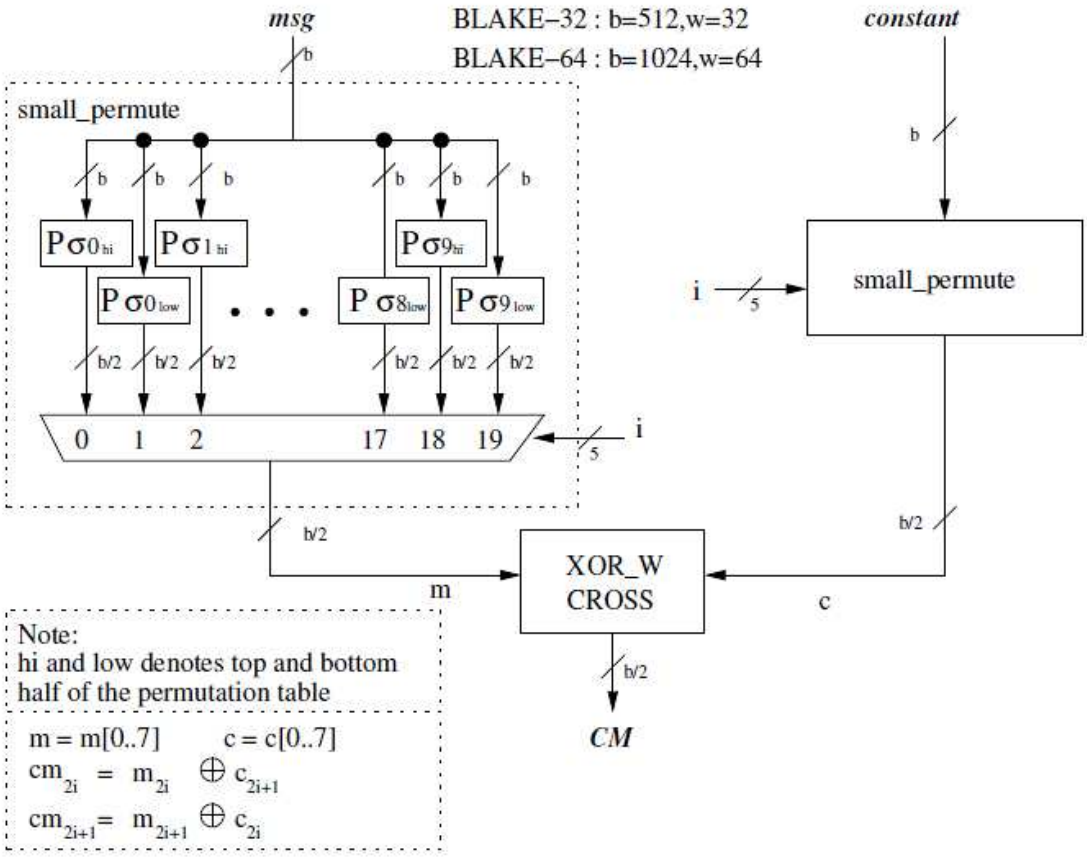


Figura 25. Permutación algoritmo BLAKE [Ref.64]

Se opera el bloque de mensaje *msg* obteniendo la cadena *m* y se realiza una pequeña permutación de la constante *c*. El nuevo valor de *m* se selecciona en función del número de ronda con un multiplexor de ancho precedido por las constantes permutaciones.

La señal se selecciona de los ciclos multiplexor de entre 0 y19 hasta terminar de ejecutar todas las rondas de BLAKE. Cada salida del multiplexor se mezcla con la respectiva constante *c* utilizando para ello la transformación XOR_W_CROSS.

Esta es una tabla de permutación:

	hi								low							
σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

Tabla 9. Tabla Permutación algoritmo BLAKE [Ref.64]

Las operaciones que se realizan dentro de la unidad del núcleo CORE se muestran en siguiente figura:

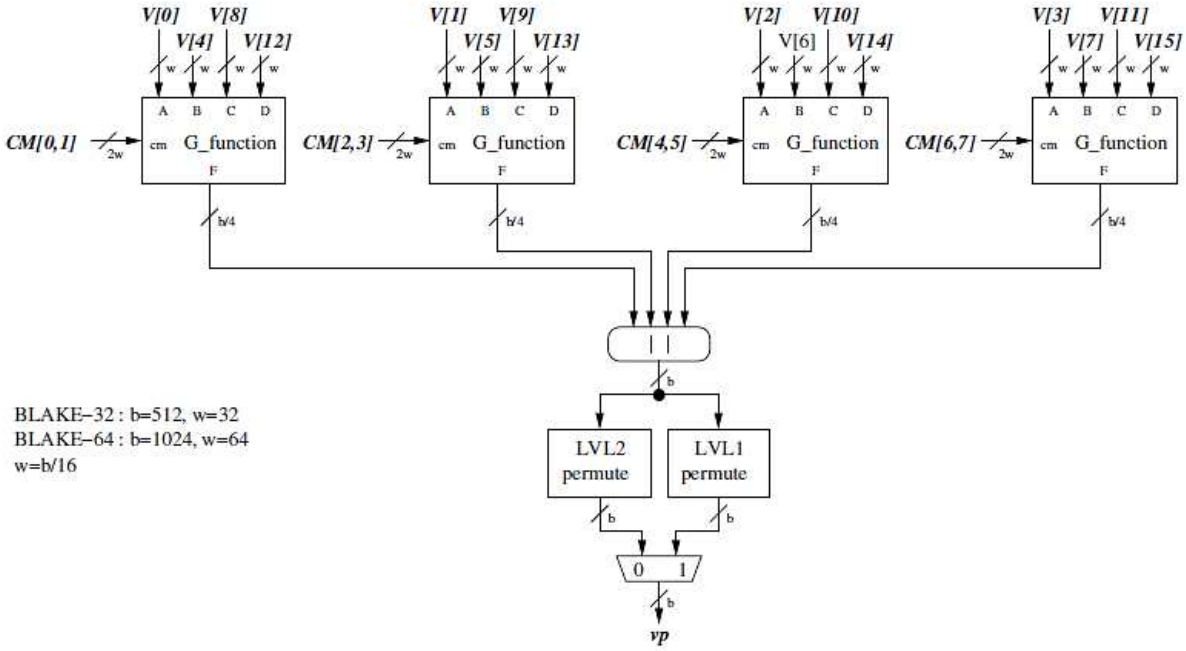


Figura 26. CORE algoritmo BLAKE [Ref.64]

Existen dos niveles para la función G por lo que se requiere una permutación entre la primera y la segunda mitad de ronda. Esta permutación se muestra en la siguiente tabla:

LVL2 (forward)

input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
output	0	1	2	3	5	6	7	4	10	11	8	9	15	12	13	14

LVL1 (inverse)

input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
output	0	1	2	3	7	4	5	6	10	11	8	9	13	14	15	12

Tabla 10. Permutación G algoritmo BLAKE

LVL2 transforma el estado de la matriz en una nueva matriz apropiada para la segunda mitad de la ronda y LVL1 es una permutación inversa a LVL2.

En la siguiente figura se muestra la función *G* del *CORE*:

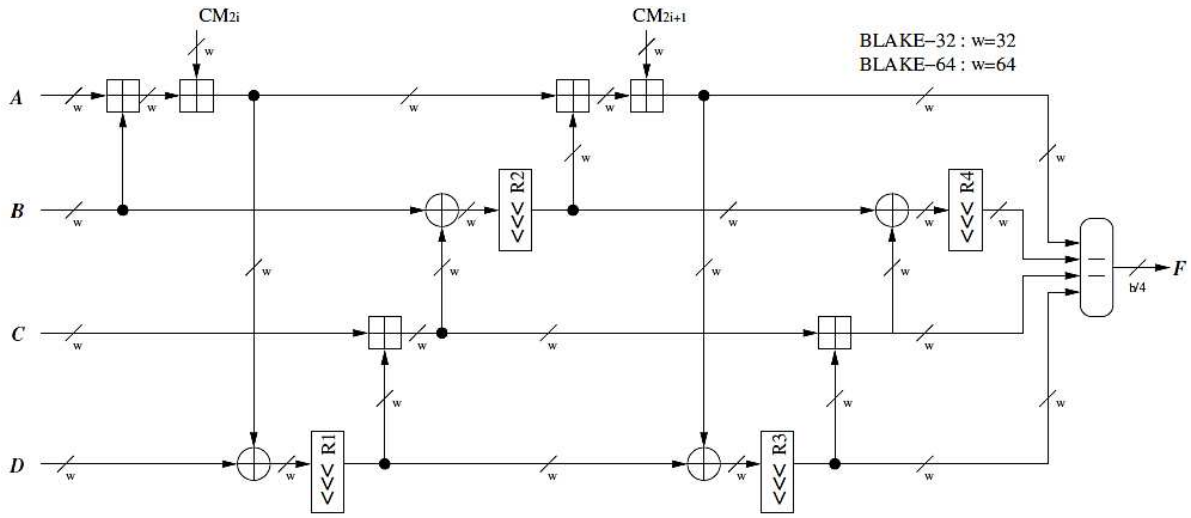


Figura 27. Función G del CORE algoritmo BLAKE [Ref.64]

La operación XOR utilizada para calcular los valores de entrada CM_{2i} y CM_{2+i} normalmente se representan como una parte de la función *G* pero en este diseño se han omitido puesto que las operaciones se situaron como parte de la unidad de permutación. Las constantes de rotación *R1, R2, R3* y *R4* están constantemente rotando y sus valores se muestran a continuación:

R1	R2	R3	R4
16	12	8	7

Tabla 11. Constantes de rotación algoritmo BLAKE-32

Algunas de las diferencias entre las dos versiones del algoritmo BLAKE-32 y BLAKE-64 son las siguientes:

- BLAKE-64 duplica el tamaño de palabra de BLAKE-32 lo cual aumenta el tamaño del bloque también haciendo que *IV* y la constante cambien de 512 bits a 1024 bits.
- BLAKE-64 presenta un incremento en el número de rondas de mezcla de 10 a 14. Este dato está obtenido, como se ha mencionado anteriormente, de la propuesta del algoritmo presentada en la segunda ronda del concurso y actualmente este valor ha pasado a ser de 16 rondas de mezcla en el caso de BLAKE-64 y de 14 rondas de mezcla en el caso de BLAKE-32. Como resultado el número de ciclos de reloj necesarios para el procesado de un único bloque de mensaje aumenta de 21 a 29.

  Todo esto hace necesario que la selección de la señal del multiplexor en la unidad de permutación se coloque de nuevo cuando el número de ronda llega a 10 y que por lo tanto después de llegar a 19 esta señal de selección vuelva a 0.
- En BLAKE-64 las constantes de rotación se ajustan para incrementar tamaño de palabra. Estos valores se describen en la siguiente tabla:

R1	R2	R3	R4
32	25	16	11

Tabla 12. Constantes de rotación algoritmo BLAKE-64

A lo largo del concurso se fueron realizando ajustes en el diseño del algoritmo para ajustarlo a los nuevos requisitos que el NIST propuso. Uno de estos cambios fue desglosar y renombrar las dos versiones existentes BLAKE-32 y BLAKE-64 en cuatro versiones llamadas BLAKE-224, BLAKE-256, BLAKE-384, y BLAKE-512 consiguiendo de esta manera diferenciar las funciones en su versión final.

Otro de los cambios realizados fue incrementar el número de rondas de 10 a 14 rondas para hashes de 224 y 256 bits y de 14 a16 rondas para hashes de 384 y 512 bits consiguiendo así algoritmos más seguros y respetando su rapidez.

4.4.3 GROESTL

Es una colección de funciones hash para mensajes de 224, 256, 384 y 512 bits desarrollada por Praveen Gauravaram, Lars Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, y Søren S. Thomsen con la intención de su presentación como participante en el concurso SHA-3.

El algoritmo utiliza la caja-S del AES para hacer una construcción personalizada y es capaz de alcanzar, según sus creadores, una velocidad de procesado de 21,4 ciclos por byte en procesadores Intel Core 2 Duo.

Su estructura está basada en permutaciones individuales, en lugar de en una gran familia de permutaciones indexadas por una clave, consiguiendo de esta manera que no haya ataques por clave, un alto rendimiento y una mayor simplicidad de la función.

En Groestl el tamaño del estado interno es significativamente mayor que el tamaño de la salida lo cual hace que los ataques genéricos conocidos no puedan encontrar con facilidad vulnerabilidades sobre esta función.

Por otro lado su diseño es distinto al diseño de los algoritmos de la familia SHA dado que, al igual que las funciones hash, este algoritmo divide la entrada en bloques y realiza los cálculos de forma iterativa manteniendo durante el proceso un estado hash al menos dos veces y que solo es truncado al final del cálculo del resumen.

En su construcción primero se opera con la función de compresión rellenando el mensaje y dividiéndolo en bloques que serán procesados secuencialmente. Después se seleccionan los valores iniciales y se procesan los bloques de mensaje de la siguiente forma:

$$hi \leftarrow f(hi-1,mi) \text{ for } i = 1,...,t.$$

Existen dos entradas que generarán dos salidas diferentes, una para cada entrada. La primera entrada es el bloque del encadenamiento y la segunda entrada es el bloque de mensaje.

Después de procesar el último bloque de mensaje la salida de la función hash será calculada de la siguiente manera:

$$H(M) = (ht)$$

El esquema de construcción de la función hash en Groestl es el siguiente:

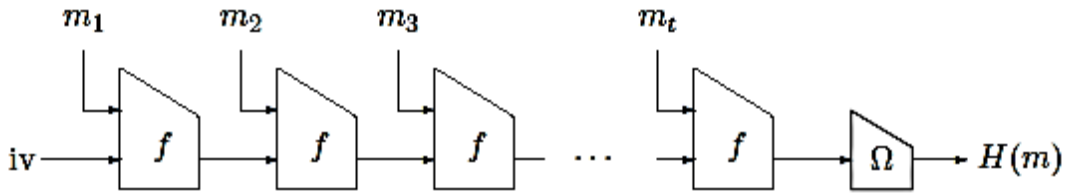


Figura 28. Función Hash algoritmo Groestl [Ref.42]

Donde Ω es una transformación de salida.

La función de compresión está basada en dos permutaciones P y Q y se define mediante la siguiente fórmula:

$$f(h,m) = P(h \oplus m) \oplus Q(m) \oplus h$$

Otra manera de visualizar la construcción de la función de compresión f es la siguiente:

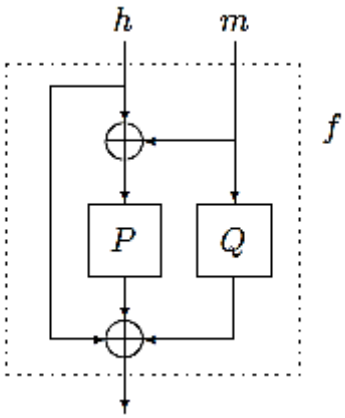


Figura 29. Función de compresión algoritmo Groestl [Ref.42]

Las funciones de permutación P y Q están diseñadas siguiendo la estructura del algoritmo Rijndael pero con la diferencia de que estas operan sobre matrices de 8x8 y de 8x16 en lugar de sobre matrices de 4x4.

Al ser P y Q mucho más grandes que el tamaño del estado en Rijndael, que es de 128 bit, la mayoría de las transformaciones se han redefinido. Las transformaciones que se realizan en este algoritmo son:

- AddRoundConstant o AddRoundKey: con claves de ronda fijas pero distintas entre P y Q.
- SubBytes: transformación en la que se utiliza la caja-S.
- ShiftBytes: transformación para ampliar y diferenciar P y Q.
- MixBytes o MixColumns: transformación para operar sobre matrices de 8x8.

Algunos de los aspectos más característicos es que las rondas son idénticas entre sí y que no se utiliza una operación final de AddRoundKey. El número recomendado de rondas para resúmenes de 512 bits es de 10 rondas y para resúmenes de 1024 bits es de 14 rondas.

A continuación se explica más en detalle el algoritmo para analizar el rendimiento de hardware sobre FPGAs:

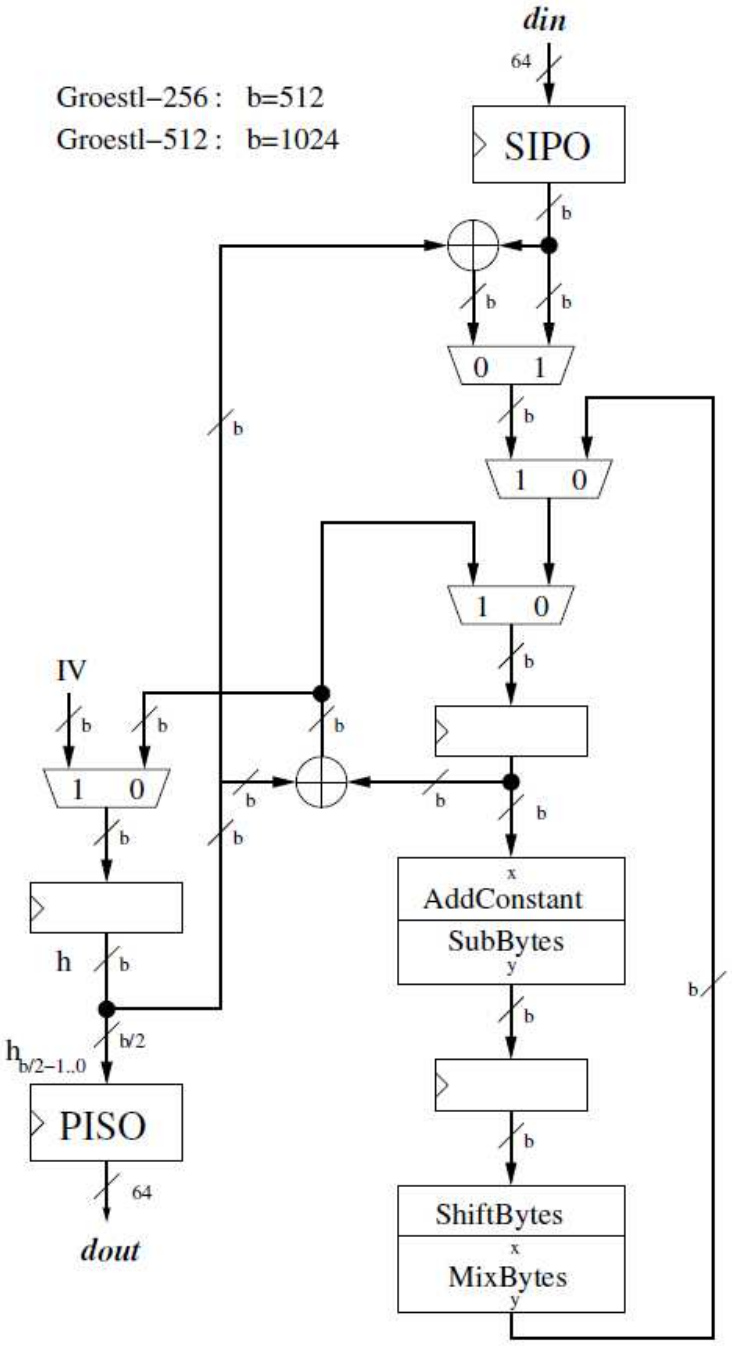


Figura 30. Diseño algoritmo GROESTL [Ref.64]

- Se aplica una arquitectura “pipeline” o tubería y su registro se insertará entre las operaciones SubBytes y ShiftBytes.
- Se aplica XOR a un bloque de mensaje con una cadena inicializada registrada para crear una entrada en la operación P en el primer ciclo de procesado.
- En el próximo ciclo se carga directamente al registro de estado una entrada mensaje como entrada a la operación Q.
- Al mismo tiempo que la primera fase de la tubería comienza a ejecutar la operación Q, la segunda etapa de la tubería continúa la ejecución de la operación P. La primera fase de la tubería consiste en la operación ADD_SUB y la segunda etapa del recorrido consiste en las operaciones de ShiftBytes y MixBytes.
- Una parte de la función P siempre se realiza un ciclo antes de la parte correspondiente de la función Q.
- Para la finalización hay dos ciclos de reloj. En el primer ciclo de reloj al valor de cadena se le aplica XOR con el valor final de P, mientras que Q se está tramitando todavía. En el segundo ciclo el resultado final de Q se mezcla con el valor de la cadena. El proceso se repitirá hasta que todos los bloques de un mensaje se mezclan a fondo.
- Por último el valor hash se tomará de la mitad inferior del valor de la cadena.

En la siguiente figura se muestra cómo se realizan las operaciones SubBytes AddConstant:

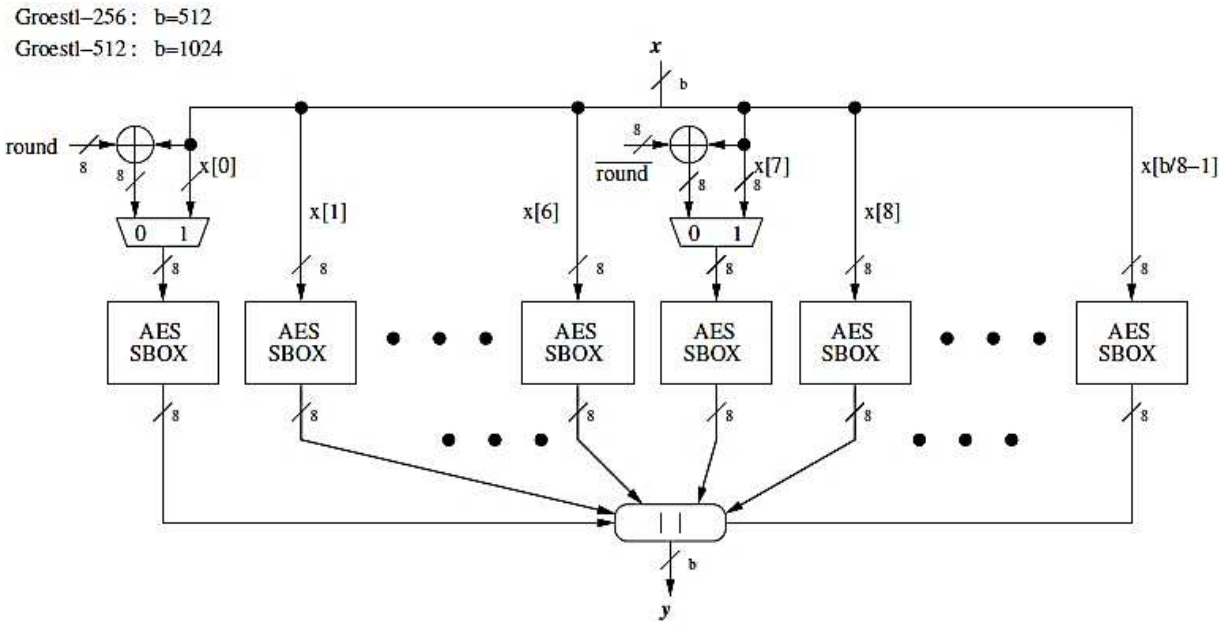


Figura 31. Operaciones SubBytes y AddConstant algoritmo GROESTL [Ref.64]

- Al número de ronda se le aplica XOR con el primer byte de mensaje en la operación P.
- En la operación Q al número de ronda se le aplica XOR con el octeto 8.
- Para finalizar todos los bytes pasan por la caja S del AES.

La operación ShiftBytes se realiza girando todos los bytes en la fila i de la derecha con σ donde σ es σ = [0, 1, 2, 3, 4, 5, 6, 7].

La caja S utilizada es la siguiente:

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 32. Caja S Groestl [Ref.41]

En la siguiente figura se muestra la estructura de la operación MixBytes:

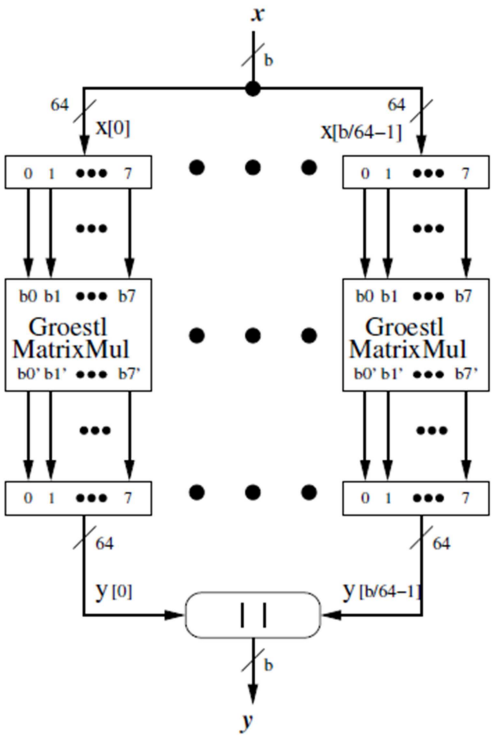


Figura 33. Operación MixBytes algoritmo GROESTL [Ref.64]

4.4.4 JH

Es una familia de funciones hash criptográficas creada por Hongjun Wu con las versiones JH-224, JH-256, JH-384 y JH-512. Tiene un estado de 1024 bits y trabaja con entradas de 512 bits.

El algoritmo utiliza 64 boques de mensaje que pasan a través de una función de compresión. El procesado de una entrada consta de 3 pasos:

- XOR del bloque de entrada en la mitad izquierda del estado.
- Una permutación sin clave en la ronda 35 que consta de 35 repeticiones en las que divide la entrada en 256 entradas de 2 bits utilizando la caja-S.
- XOR del bloque de entrada en la mitad derecha del estado.

A continuación se detalla más en profundidad la estructura de la función empezando para analizar el rendimiento de hardware sobre FPGAs:

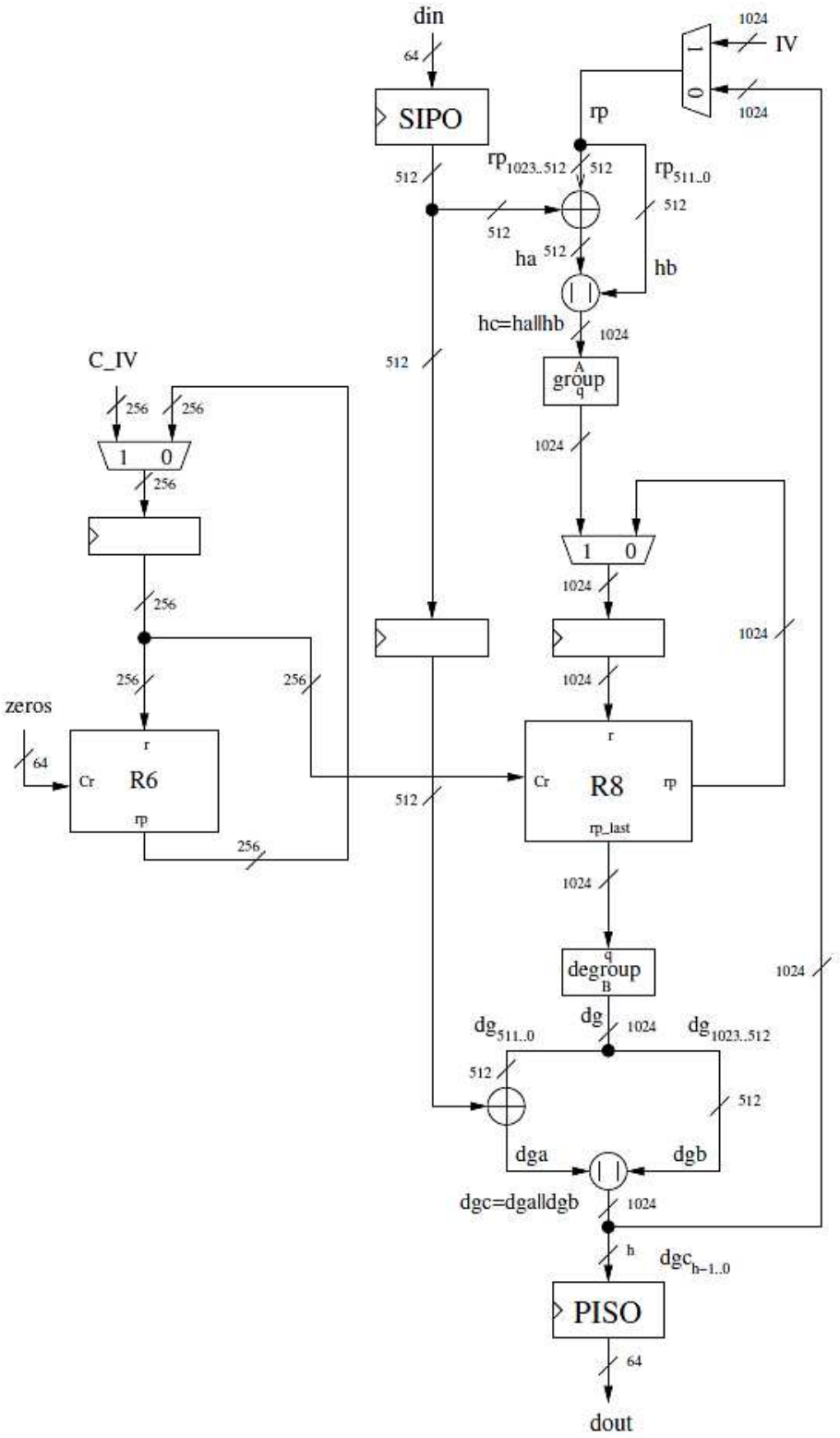


Figura 34. Diseño algoritmo JH [Ref.64]

Al principio los tres registros que aparecen en el diagrama se inicializan con la variable *C-IV* que es el primer bloque de mensaje. El estado inicial depende del valor de *IV* y del primer bloque de mensaje.

El estado interno se transforma con la fase de transformación *R8* durante 36 rondas en las cuales en cada una de ellas se genera una constante diferente *Cr* utilizando para ello la transformación *R6*.

Una vez terminado el proceso la salida de *R8* es desagrupada y su mitad inferior se mezcla con el bloque de mensaje almacenado en el registro temporal para así crear un nuevo valor *DGC*. Si hay más bloque de mensaje la mitad superior del valor de la cadena se mezcla con el siguiente bloque y el resultado se procesa a través del grupo de transformación. Los pasos anteriores se repiten hasta que todos los bloques de mensajes están procesados El valor del hash será el resultado la cadena que produzca el último bloque procesado.

En la siguiente imagen se muestran las permutaciones de agrupación y des-agrupación:

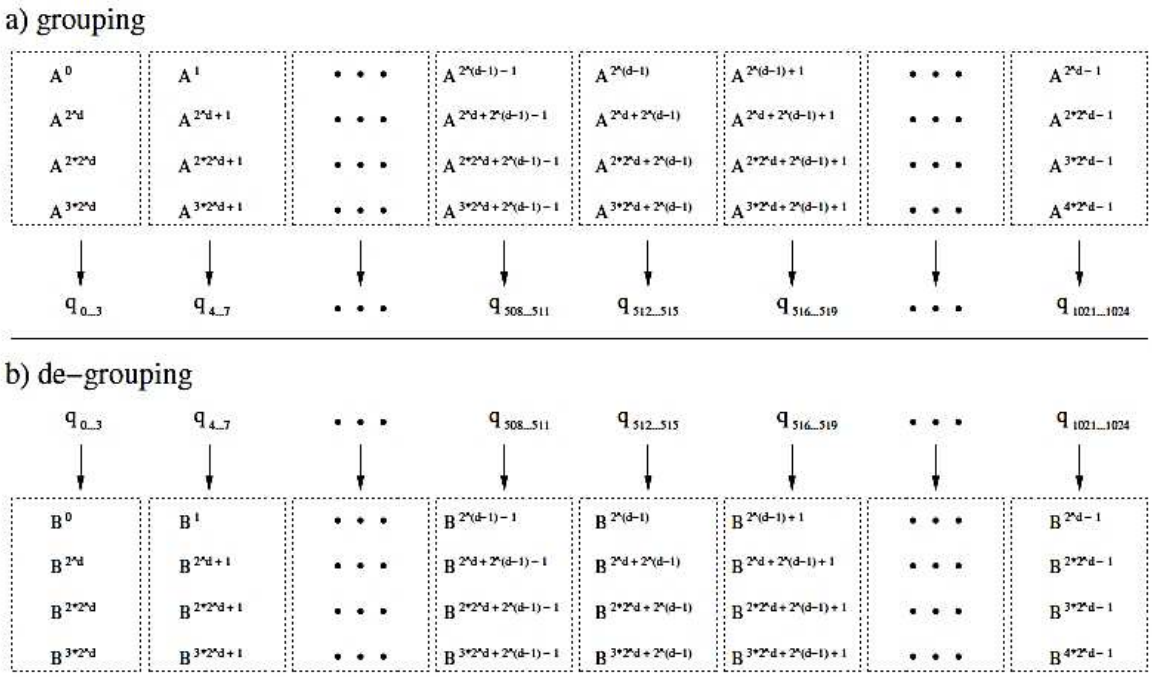


Figura 35. Agrupación y Des-agrupación algoritmo JH [Ref.64]

En la siguiente figura se muestra una ronda de este algoritmo:

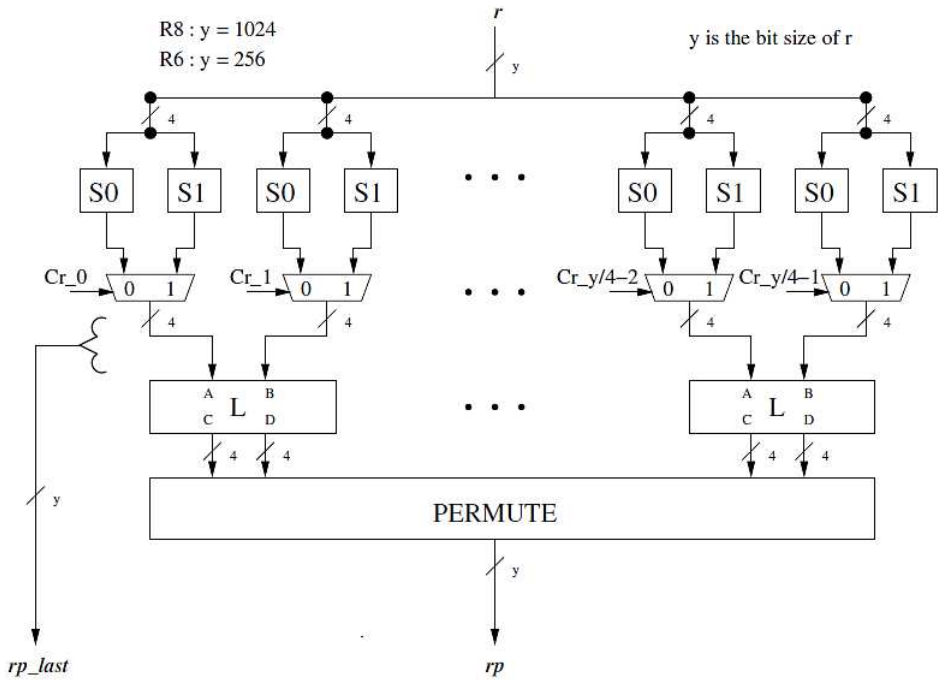


Figura 36. Ronda algoritmo JH [Ref.64]

Los resultados de la caja-S son seleccionados por el bit correspondiente de la ronda *Cr*. La caja-S es la siguiente:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_0[x]	9	0	4	11	13	12	3	15	1	10	2	6	7	5	8	14
S_1[x]	3	12	5	13	5	7	1	9	15	2	0	4	11	10	14	8

Tabla 13. Caja S algoritmo JH

Dos salidas consecutivas forman una entrada a la unidad de transformación lineal *L*:

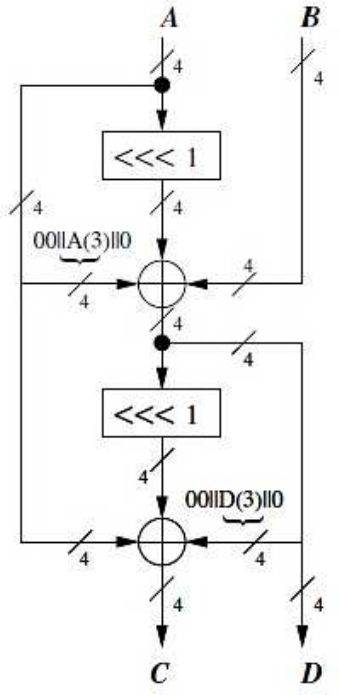


Figura 37. Formación entrada a L algoritmo JH [Ref.64]

Las salidas permutadas se transforman más tarde por la unidad de permutación.

Todas las operaciones son las mismas para ambas variantes. La única excepción es la selección de la función de salida en JH-512 donde se seleccionan 512 bits de la cadena de valor en lugar de 256 bits, que es lo que ocurre en el algoritmo JH-256.

4.4.4 SKEIN (MADEJA)

Es una función hash criptográfica creada por Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas y Jesse Walter.

Más que una función de hash se trata de una familia completa de algoritmos de hash conservadores y con un diseño modular que facilitan el criptoanálisis de dominio público. Para garantizar la seguridad esta familia de funciones está basada en el cifrador de bloques Treefish, consiguiendo rapidez a pesar de ello.

La elección de la clave Treefish está inspirada en la elección de clave Skipjack, en la que la clave es del mismo tamaño que el texto. El valor del tweak es de 128 bits y cada subclave está compuesta utilizando 3 sumas. Para evitar ataques existe un contador.

Este algoritmo consta solamente de 3 operaciones primitivas, de una función de compresión sencilla y de unos pasos posteriores que se basan en la repetición de dicha función. Los estados internos pueden ser de 256, 512 y 1024 bits y la salida puede ser de tamaños arbitrarios.

- Skein-512 es la propuesta principal y puede ser utilizada con seguridad para todas las aplicaciones de hash actuales y previsiblemente en un futuro.
- Skein-1024 es la variante ultraconservadora y debido a que tiene dos veces el tamaño interno de 512 si un futuro ataque logra romper Skein-512 probablemente Skein-1024 siga siendo seguro. Esta variante además se puede ejecutar casi dos veces más rápido.
- Skein-256 es la variante de poca memoria, que se puede implementar utilizando aproximadamente 100 bytes de RAM simplemente.

Respecto a la rapidez, los autores afirman que es capaz de procesar a la velocidad 6,1 ciclos por byte en una CPU Intel Corre 2 en modo de 64 bits, lo que significa que es el doble de rápido que SHA- 512 y el triple de rápido que SHA-256, por lo que se pueden procesar hasta 500 MBytes/s por cada núcleo en un procesador Core2 Duo, de 64 bits y 3,1 Ghz.

Esta familia de algoritmos se puede utilizar con distintas funciones como pueden ser funciones de derivación de claves, cifradores de flujo, funciones de autenticación sin sobrecarga, etc, haciendo simplemente algunos cambios sobre ella.

Tiene un factor de seguridad de 2,9 que además sus diseñadores constatan que tiene basándose en propiedades que lo demuestran.

Por otro lado se puede utilizar en una gran variedad de plataformas tanto software como hardware haciendo simplemente algunas modificaciones sobre ella, siendo que cuanto más grande sea la plataforma más grande tendrá que ser el algoritmo para asegurar la velocidad requerida.

A continuación se muestra parte de su estructura:

Unico Bloque de iteración: modo de encadenamiento que utiliza el algoritmo Threefish para construir una función de compresión que asigne un tamaño de entrada arbitrario a un tamaño de salida fijo.

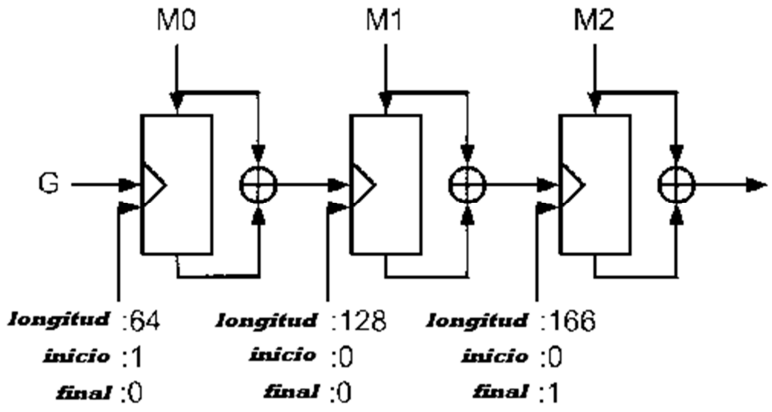


Figura 38. Cifrando tres bloques de mensaje utilizando Bloque Único de Iteración algoritmo Skein [Ref.57]

Los bloques de mensaje M0 y M1 contienen 64 bytes de datos de cada uno y M2, que es el bloque final, contiene 38 bytes de datos. El valor tweak para cada bloque codifica los bytes se hayan procesado hasta el momento además de si esta es la primera o última secuencia de la computación del bloque único de iteración. El tweak también codifica un tipo que se utiliza para distinguir los diferentes usos del modo bloque único de iteración.

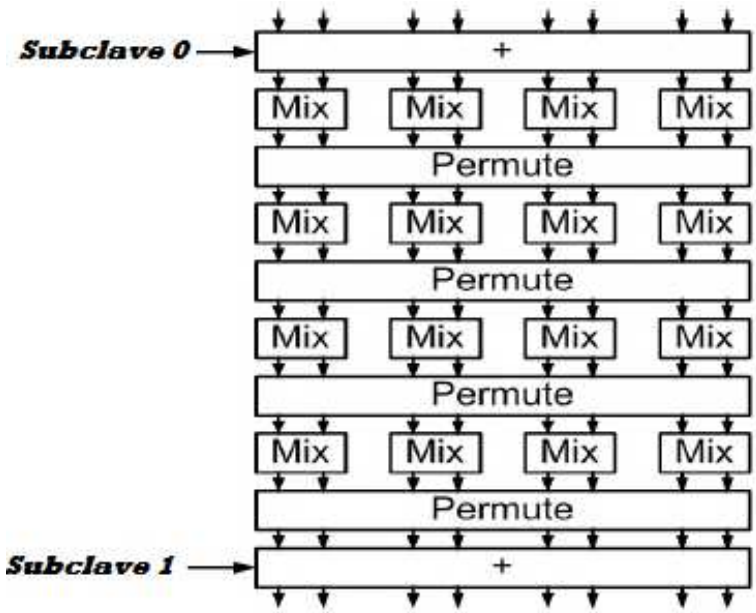


Figura 39. Cuatro de las 72 rondas del cifrador de bloque Threefish-512 algoritmo Skein [Ref.57]

A continuación se describe un poco más en profundidad el algoritmo para analizar el rendimiento de hardware sobre FPGAs:

En la siguiente figura se muestra el camino que realizan los datos:

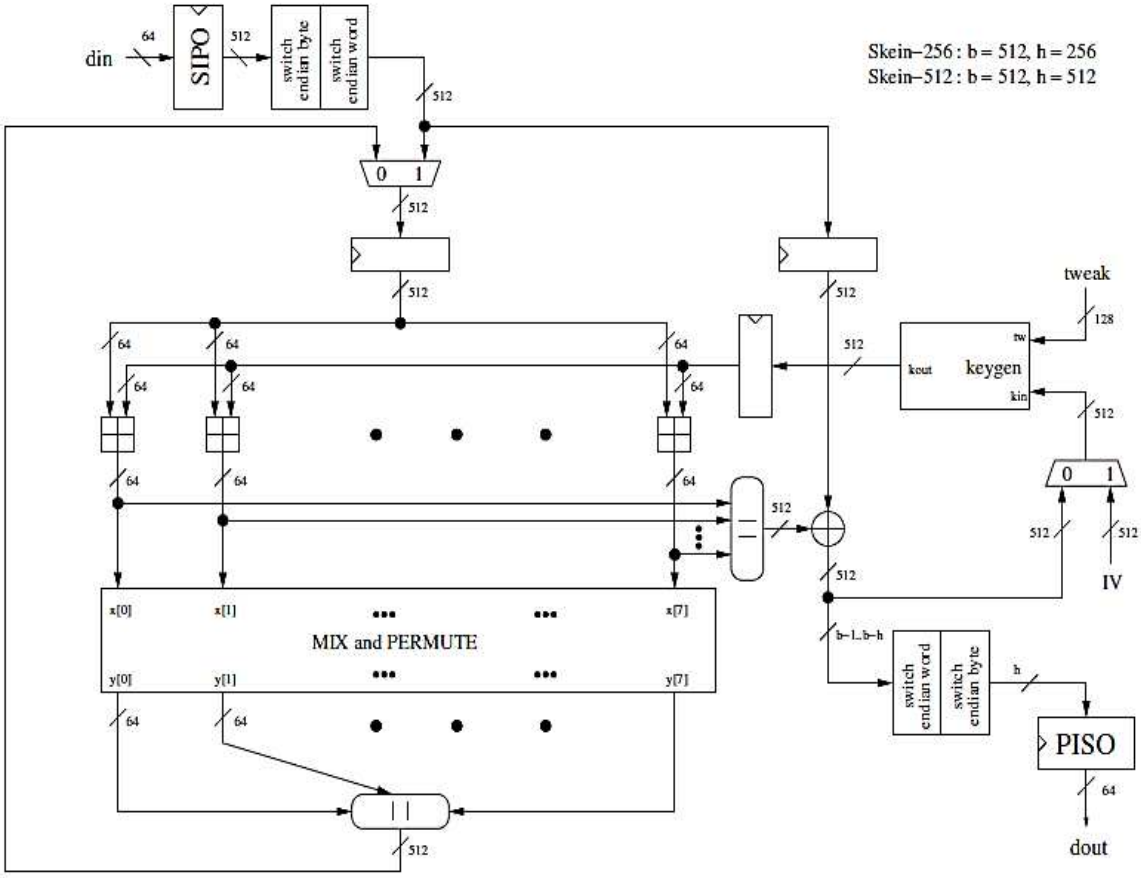


Figura 40. Diseño algoritmo Skein [Ref.64]

El camino de datos de este algoritmo se puede separar en dos partes principales: la generación de claves y la "*Ronda Madeja*". La ronda incluye una capa de adiciones de 64 bits y 4 veces el desenrollo de la mezcla y la permutación. Para inicializar el estado interno se utiliza un bloque de mensaje con es una serie de ocho palabras de 64 bits. Una vez cada cuatro rondas de mezcla y se permuta agregando al estado una

subclave por cada bloque de mensaje. Dependiendo de la versión del cifrador por bloques hay un número u otro de rondas siendo para la versión de 256 bits de 72 rondas, para la versión de 512 bits de 72 rondas y para para la versión de 1024 bits de 80 rondas. Cada ronda es muy simple y fácil de analizar pero la complejidad radica en el número de ellas. La difusión completa tarda entre 9 y 11 rondas.

Debido a los 4 desenrollos estas rondas se ejecutan en 18 ciclos de reloj y la despenalización se realiza después de la última ronda que se ejecuta, justo al final de cada bloque de mensaje procesado con el fin de generar una nueva cadena de valor. Esta operación es equivalente a una adición entre el estado y la clave seguido por un XOR con el bloque de mensaje actual.

La unidad de generación de claves tiene dos fuentes de entrada, el valor de la cadena y el ajuste. El valor de cadena se utiliza en la unidad de generación de claves y se calcula a partir del bloque de mensaje anterior o tomando como un vector de inicialización al principio del mensaje.

En la siguiente figura se muestran la generación de una de las claves la cual es generada para cada bloque de mensaje nuevo:

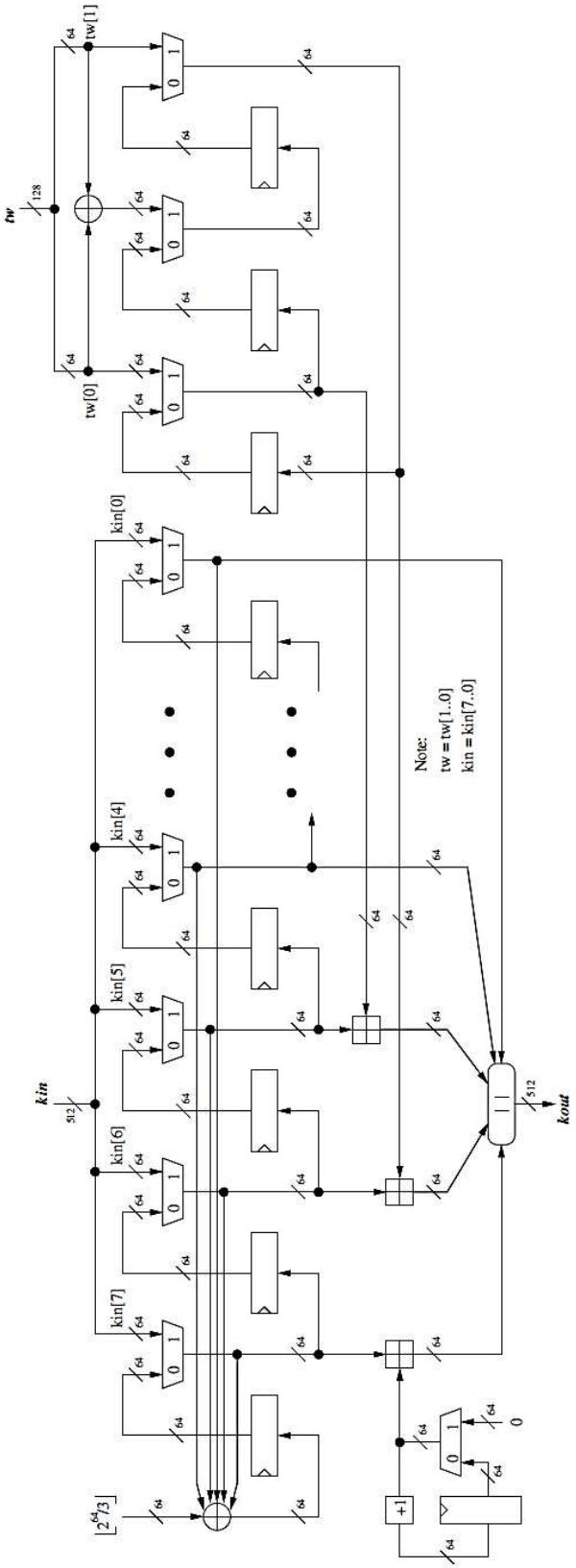


Figura 41. Generación de clave algoritmo Skein [Ref.64]

En la siguiente figura se muestra uno de los cuatro desenrollos de la mezcla y la permutación:

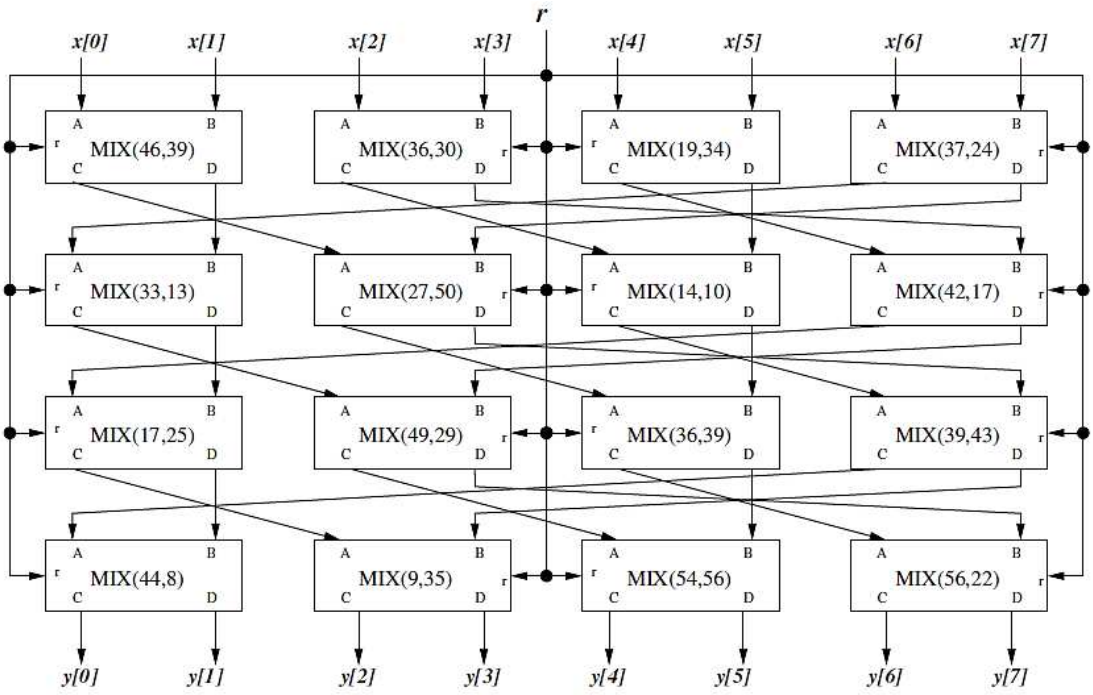


Figura 42. Desenrollos de mezcla y permutación algoritmo Skein [Ref.64]

Esta unidad está basada en 16 instancias de la operación mezcla, la cual se muestra en la siguiente figura:

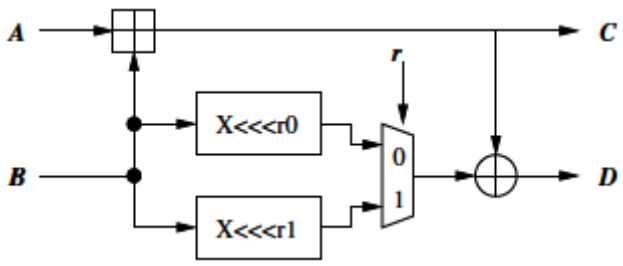


Figura 43. Operación mezcla algoritmo Skein [Ref.64]

Las constantes de rotación se presentan en la siguiente tabla y el número de ronda es calculado módulo 8:

N_w	8			
j	0	1	2	3
0	46	36	19	37
1	33	27	14	42
2	17	49	36	39
3	44	9	54	56
4	39	30	34	24
5	13	50	10	17
6	25	29	39	43
7	8	35	56	22

Tabla 14. Constantes de rotación algoritmo Skein

La permutación ejecutada entre cada ronda de mezcla es mostrada en la siguiente figura:

x	0	1	2	3	4	5	6	7
y	2	1	4	7	6	5	0	3

Tabla 15. Permutación algoritmo Skein

4.5 KECCAK, EL GANADOR

Es una función hash criptográfica creada por Guido Bertoni, Joan Daemen, Michaël Peeters y Gilles Van Assche.

Este algoritmo se puede tomar como sucesor del algoritmo RadioGatún, el cual fue desarrollado a su vez para mejorar la función de hash Panamá aunque con filosofías de diseño diferentes siendo que esta última podía ser utilizada como función hash o como cifrador de flujo. RadioGatún es el algoritmo contra-opuesto a la construcción Merkle-Damgard puesto que este RadioGatún se centra en una función de compresión de longitud variable con la aplicación iterativa de una sola función de ronda sobre el estado.

RADIOGATÚN

Para poder entender mejor la estructura de Keccak son necesarias unas nociones básicas sobre el diseño de su predecesor RadioGatún.

En RadioGatún el estado se divide en dos partes llamadas cinturón o *belt function* y molino o *mill function*.

En la siguiente imagen se muestras estas dos partes:

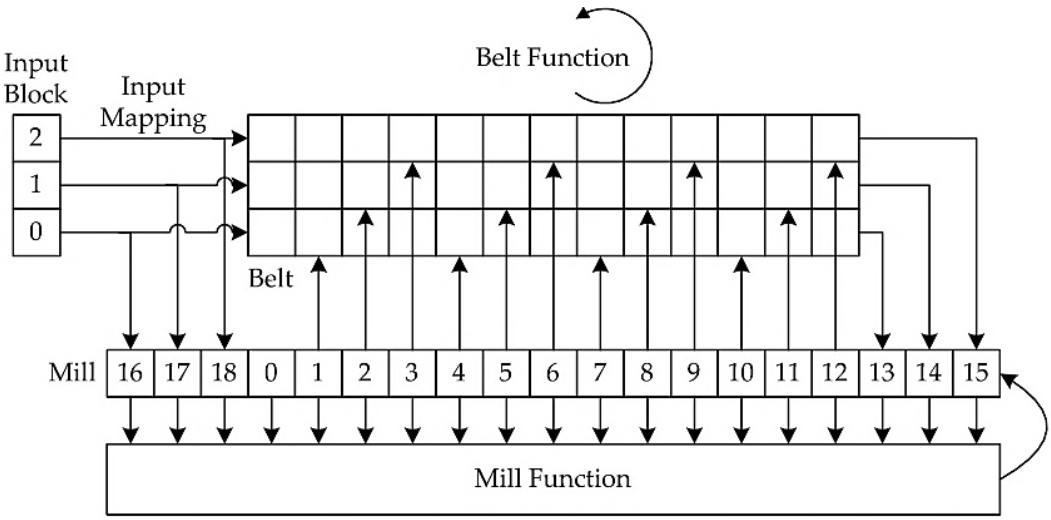


Figura 44. RadioGatún [Ref.92]

- Las operaciones utilizadas a nivel de bits son las operaciones lógicas AND, XOR, NOT y cambios cíclicos en las palabras.
- Se puede tomar cualquier tamaño de palabra que esté comprendido entre 1 y 64. La opción de 64 bits es la opción por defecto y es ideal para plataformas de 64 bits pero por otro lado para plataformas de 32 bits la mejor opción son palabras de 32 bits. La seguridad vendrá determinada por el tamaño de palabra que se elegida.

Una de las ventajas de este modelo es el buen rendimiento que proporciona con SHA-1 ya que es muy competitivo, compacto y rápido.

Podemos diferenciarlo de las funciones esponja en que en la construcción esponja la entrada se aplica a la salida y se extrae de la misma parte del estado y esto no ocurre en RadioGatún, y por otro lado en que en RadioGatún hay un número de rondas en blanco entre la aplicación de la entrada y la extracción de la salida y en una función esponja esto no ocurre.

Volviendo a Keccak y a sus diferencias con respecto a RadioGatún, éste último la transformación que aplica al estado entre la inserción de bloques de entrada o la extracción de los bloques de salida es simplemente una función de ronda que no pretende estar libre de las propiedades estructurales que mencionábamos anteriormente, además que como ya hemos dicho anteriormente, RadioGatún no es una función esponja dado que no sigue este tipo de construcción.

La función de permutación en Keccak consiste en la iteración de una función de redondeo simple, similar a un cifrado de bloques y las operaciones que se realizan bit a bit son XOR, AND, NOT y rotaciones y además no hay necesidad de listar las búsquedas, operaciones aritméticas, o datos que dependen de rotaciones.

En cuanto a la eficiencia de Keccak, sus autores afirman que es capaz de procesar a la velocidad 12,5 ciclos por byte en una CPU Intel Core 2. El software toma alrededor de 13 ciclos por byte en la plataforma de referencia definido por el NIST. En cuanto al hardware, es rápido y compacto.

Este algoritmo puede utilizarse con una clave y además puede generar salidas infinitas, por lo que es adecuado como cifrador de flujo.

A continuación se detallará más a fondo el algoritmo mediante unas figuras:

Para el modelo Keccak-256, el bloque de mensaje de entrada tiene el tamaño de 1088 bits y el camino de los datos se muestra que aparece a continuación:

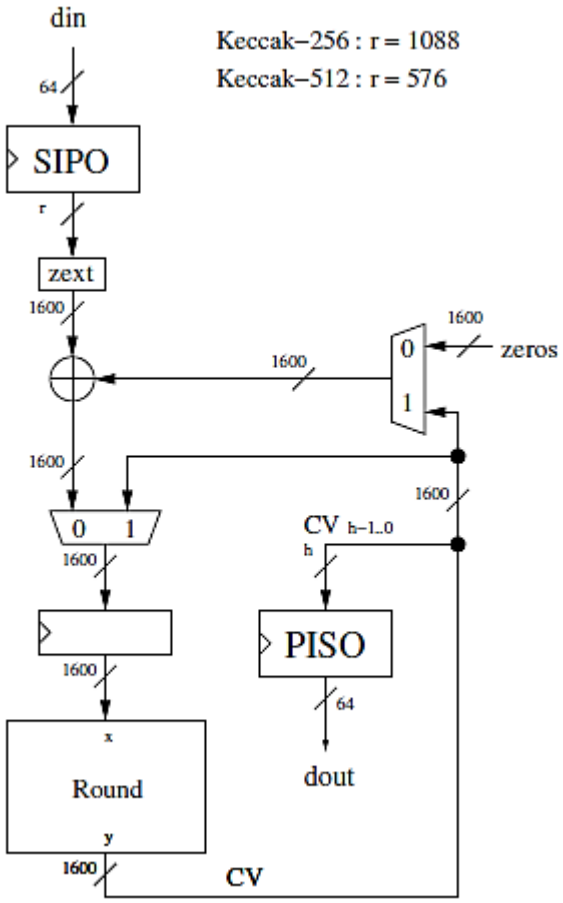


Figura 45. Diseño algoritmo Keccak [Ref.64]

En esta figura por cada bloque de mensaje se pone una entrada con cero-extendido para producir un estado de 1600 bits.

Este estado puede ser visto como una matriz de palabras de 5x5 de 64 bits como se muestra la siguiente figura.

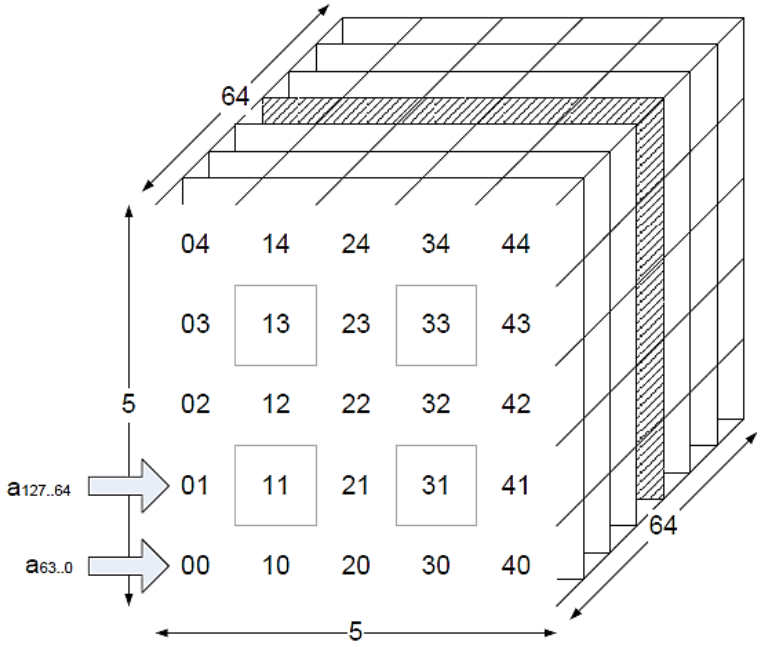


Figura 46. Matriz estado algoritmo Keccak [Ref.64]

En una entrada extendida se hace un XOR con el valor de cadena, y para el primer bloque de mensaje, el valor de cadena es cero. Después el estado se transforma utilizando "Ronda Keccak" durante 24 rondas. Por último, para el valor del hash se selecciona el valor de la cadena del último bloque de mensaje.

En las siguientes dos figuras se muestran las descripciones de la "Ronda Keccak".

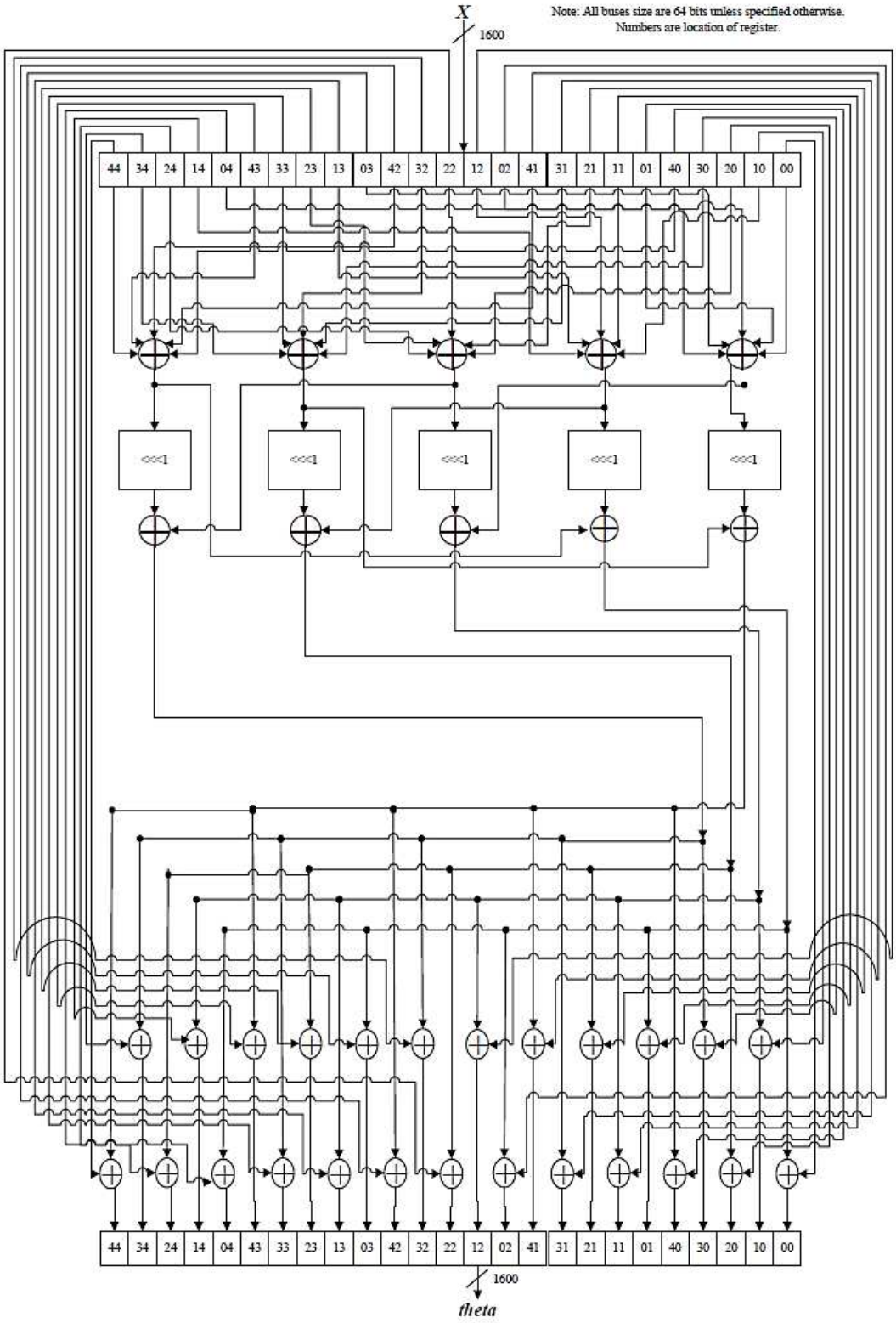


Figura 47. Descripción Ronda algoritmo Keccak 1[Ref.64]

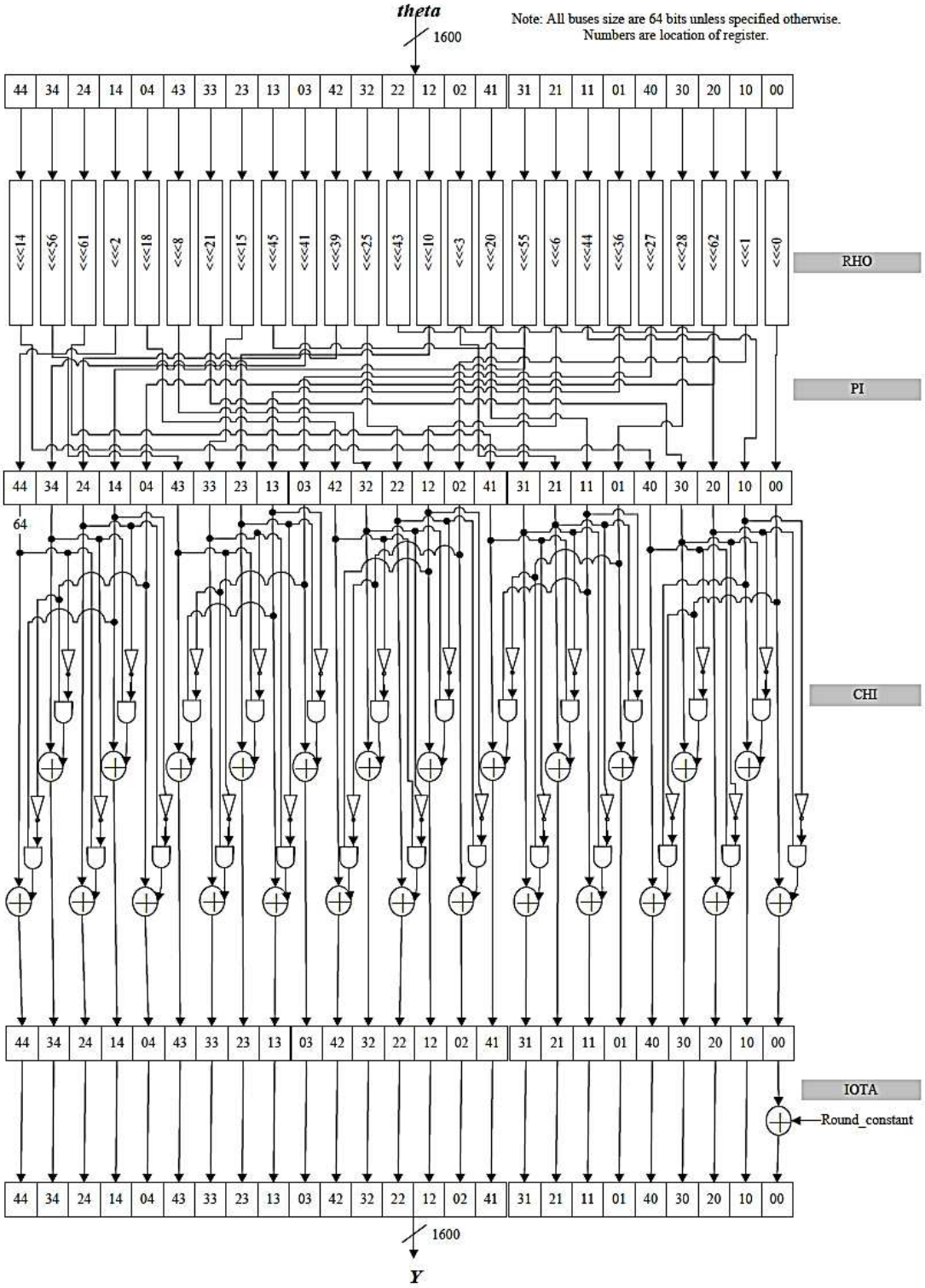


Figura 48. Descripción Ronda algoritmo Keccak 2[Ref.64]

Todas las operaciones son las mismas para ambas variantes. La única excepción es que para la salida de Keccak-512 se seleccionan 512 bits de la cadena de valor en lugar de 256 bits, que es lo que ocurre en el algoritmo Keccak-256.

4.6 Gráfico comparativo de los 5 finalistas

En este gráfico se muestra una comparativa de los algoritmos y volver a remarcar que los datos son obtenidos antes de octubre de 2010 que fue cuando se seleccionaron los 5 finalistas: Los parámetros principales de las dos variantes de las funciones se describen a continuación:

ALGORTIMO	**PRINCIPALES TAREAS ITERATIVAS**	**PRINCIPALES OPERACIONES**	**Variante de 256 bits**				**Variante de 512 bits**				**Primera aproximación**	**Rendimiento Previsto**
			Tamaño Estado	*Tamaño Bloque*	*Numero Rondas*	*Tamaño Palabra*	*Tamaño Estado*	*Tamaño Bloque*	*Numero Rondas*	*Tamaño Palabra*		
BLAKE	Gi…Gi+3	mADD3, ADD, XOR	512	512	10	32	1024	1024	14	64	2	1.43
GROESTL	Ronda modificada de AES	S-box AES 8x8, GF MUL x02 y x03, XOR	512	512	10	64	1024	1024	14	64	2	1.43
JH	Ronda Función Rs	S-box 4x4, GF MUL x2 y x5, XOR	1024	512	36	64	1024	512	36	64	1	1
KECCAK	Ronda R	NOT, AND, XOR	1600	1088	24	64	1600	576	24	64	1	0.53
SKEIN	4 Rondas de Threefish-512	ADD, XOR	512	512	72	64	512	512	72	64	1	1

Tabla 16. Grafico comparativo 5 finalistas

En esta tabla me muestra una comparativa respecto al rendimiento que se ha observado en los 5 finalistas, antes de su elección:

	Variante de 256 bits			**Variante de 512 bits**		
Función	*Entrada/Salida bus de datos*	*F. Tiempo Hash (ciclos reloj)*	*Rendimiento (Mbit/s)*	*Entrada/Salida bus de datos*	*F. Tiempo Hash (ciclos reloj)*	*Rendimiento (Mbit/s)*
BLAKE	64	2+8+21*N+4	512/(21*T)	64	2+16+29*N+8	1024/(29*T)
Groestl	64	3+8+21*N+4	512/(21*T)	64	3+16+29*N+8	1024/(29*T)
JH	64	3+8+36*N+4	512/(36*T)	64	3+8+36*N+8	512/(36*T)
Keccak	64	3+17+24*N+4	1088/(24*T)	64	3+9+36*N+8	576/(24*T)
Skein	64	2+8+19*N+4	512/(19*T)	64	2+8+19*N+8	512/(19*T)

Tabla 17. Tabla comparativa rendimiento 5 finalistas

4. 7 OPINIONES GENERALES DEL CONCURSO

Como ha quedado claro en la exposición de las fases y del concurso en general la comunidad criptoanalista ha sido participe durante toda la trayectoria de la competición mediante la aportación de comentarios e ideas y de estudios de los algoritmos presentados al concurso.

Entre los comentarios encontrados en foros criptográficos y en material criptográfico en general se ha podido observar que durante las dos primeras fases del concurso gran parte de la comunidad criptoanalista optaba por el algoritmo BMW como algoritmo ganador de la competición. Este algoritmo finalmente no fue elegido por el NIST como participante en la tercera y última ronda del concurso por considerarlo algoritmo no "todo terreno" a pesar de superar todas las pruebas que le habían sido propuestas durante su candidatura en el concurso y de su diseño innovador.

Durante la tercera ronda se pudo observar que la idea generalizada dentro de la comunidad criptoanalista era la victoria en la competición del algoritmo de la madeja o también llamado "SKEIN" a pesar de sus muchos detractores que lo calificaban de muy conservador. Estos detractores llegaron incluso a realizar un estudio informal y personal con el objetivo de demostrar su lentitud. Como conclusión de este estudio se determinó que este algoritmo era igual de lento que el "algoritmo infierno", algoritmo que ya había sido descalificado y que simplemente era solo 2 o 3 veces más rápido que SHA-2.

Entre los defensores del concurso también existía otro pensamiento pluralizado sobre la elección de los algoritmos en las distintas fases. Opinaban que algunos de los mejores algoritmos no habían sido seleccionados por criterios de diversidad cuando éste era uno de los requisitos fundamentales para poder presentar las candidaturas al concurso. Expresaron su descontento puesto que la realización un algoritmo de resumen de este nivel suponía un esfuerzo muy grande además de muchísima dedicación.

El método de elección de los algoritmos durante las distintas fases fue muy criticado por seguidores y defensores del concurso que expresaron su malestar sobre ciertas decisiones tomadas por el NIST. Estos hicieron saber su descontento ante el método que se había estado siguiendo en el cual no todos los algoritmos debían pasaban por las mismas pruebas lo que hacía que algunos algoritmos hubieran tenido que pasar por numerosas pruebas de seguridad muy exhaustivas y que algunos de los algoritmos finalistas precisamente no. También hubo quejas respecto a los plazos de tiempos que se habían establecido para las determinadas fases calificándolos de excesivos y opinando que únicamente retrasaban el proceso de selección del algoritmo. Ante estas críticas el NIST declaró su intención de modificar las bases del concurso en el momento que fuera necesario al ser ellos la institución que había promovido el concurso y por tanto los responsables de la toma de decisiones.

La solución que idearon algunos de estos seguidores del concurso ante las injusticias que según ellos estaban ocurriendo fue proponer la elección de 8 finalistas en lugar de 5 finalistas, y de entre ellos elegir 3 algoritmos ganadores en lugar de 1. Esta idea fue rechazada puesto que a pesar del gran nivel de los candidatos el objetivo del concurso desde un principio fue encontrar el mejor algoritmo de entre todos ellos y que pudiera ser utilizado como estándar de resumen a nivel mundial.

Para algunos de los miembros de la comunidad criptoanalista la elección del algoritmo ganador no fue del todo transparente dado que los informes oficiales con los criterios que se habían seguido en las elecciones no fueron publicados a tiempo o incluso no se llegaron a publicar.

El transcurso del concurso fue muy criticado y comentado por los mismos concursantes dado que estos serían los futuros consumidores del producto SHA-3 y por lo que se vieron con derecho a realizar críticas constructivas durante toda la competición aunque estas críticas no siempre fueron constructivas.

En ciertos momentos del concurso las diferencias entre los miembros de la comunidad criptoanalista se hicieron muy notables llegando incluso a descalificarse unos a otros. Gente que se denominaban a sí mismos “de la industria” y que estaban en contra del concurso y de lo que representaba en general, insultaron a los defensores del concurso con descalificaciones como “llorones”, “seguidores con ojos cerrados”, y “poco profesionales”. Estas personas dejaron claro en todo momento su descontento con el concurso y promulgaron la no utilización del algoritmo fuera cual fuera el ganador.

Por otro lado otra parte de la comunidad criptoanalista dictaminó el fracaso del algoritmo SHA-3 al considerar que las mejoras que éste pudiera ofrecer, fuera cual fuera el algoritmo elegido, distarían muy poco de las posibilidades que ya ofrecía el algoritmo SHA-2. Durante la competición este grupo de personas empezó a planear futuros concursos e incluso durante el transcurso de la segunda ronda hicieron estudios para comprobar la eficacia y la eficiencia de los algoritmos de esta fase y demostrar que estos no eran tan seguros como en un principio parecían ser. Por este motivo empezaron a reclamar un nuevo algoritmo SHA-5 que permitiera renovar el algoritmo que aún no se había elegido.

Se opinó abiertamente sobre la necesidad de la industria de utilizar el algoritmo SHA-3. Determinaron que si se obliga a la industria a utilizar el nuevo algoritmo se estarían deteniendo los nuevos avances y las nuevas tecnologías que se pudieran desarrollar. También alegaron que los ataques que se pudieran hacer sobre los productos existentes eran impredecibles y que al estar sobradamente comprobado que los algoritmos SHA-1 y SHA-2 funcionaban en la mayoría de los casos no habría motivos para cambiarlos por el algoritmo SHA-3 siendo que éste estaría simplemente probado frente ataques teóricos y no frente ataques reales al no haber estado todavía implantado.

A pesar de la decisión por parte de la industria de seguir utilizando SHA-2 y de esperar a un futuro algoritmo SHA-4 o SHA-5 para sustituirlo, Estados Unidos dejo clara su intención de utilizar el nuevo algoritmo SHA-3.

A pesar de toda la rivalidad existente entre los participantes, defensores y detractores de los algoritmos y de la comunidad criptoanalista en general, durante todo el concurso se veló por conseguir el objetivo principal del concurso de encontrar un algoritmo seguro y rápido que pudiera ser utilizado de estándar y todos colaboraron para alcanzar esta meta. Por todo esto los competidores aceptaron de buena manera la descalificación de sus algoritmos durante las distintas fases del mismo siendo conscientes de que a pesar de haber algoritmos muy buenos solo podía haber un algoritmo ganador y éste debía ser el más completo.

La competencia y la rivalidad en todo tipo de concursos es normal y sobre todo en éste dada la importancia del concurso a nivel internacional y la importancia del producto que se desea obtener. Hay una cita de uno de los seguidores del foro de la página oficial que resume de una manera objetiva las rivalidades existentes en el concurso:

"Si la industria es un océano, nuevas funciones hash son barcos de vela con una brisa muy suave. No hay un viento que sople lo suficientemente fuerte (en la seguridad, dimensiones o parámetros de rendimiento) para hacer distinciones enormes entre los candidatos. Sin el drama, los espectadores no estarían emocionados."

4. 8 OPINIONES DE CRIPTOANALISTAS ACERCA DE LA FUNCIÓN GANADORA

El anuncio de NIST del algoritmo elegido como ganador del concurso SHA-3 decía:

"*NIST escogió Keccak de entre los otros cuatro excelentes finalistas por su diseño elegante, su amplio margen de seguridad, un buen rendimiento en general, una excelente eficiencia en las implementaciones de hardware además de por su flexibilidad. Por otro lado, Keccak complementa la familia de algoritmos SHA-2. NIST mantiene su confianza en la seguridad de SHA-2 que ahora está ampliamente implementado, y los algoritmos hash SHA-2 continuarán siendo utilizados en el futuro inmediato, como se indica en la declaración de política de hash NIST. Una de las ventajas que ofrece Keccak como el ganador de SHA-3 es su diferencia en el diseño. Parece muy poco probable que un solo ataque pueda poner en peligro los dos algoritmos…*" *[Ref. 86]*

Dicho esto, algunos criptoanalistas opinan que el NIST no eligió Keccak por tener un mejor rendimiento en general que SHA-2, dado que piensan que no es cierto, o por ser un algoritmo más seguro que SHA-2, lo cual niegan también, sino que el NIST escogió Keccak porque su diseño de construcción esponja lo hace muy diferente del diseño de SHA-2 basado en la construcción Merkle-Damgard, teniendo así un seguro en el caso de que se vulnere SHA-2.

Hay opiniones acerca de la función resultante de todo tipo, tanto a favor como en contra. Respecto a su diseño y rendimiento, opiniones dicen que el rendimiento del software de la función no es bueno y que por contrapartida el de Blake tanto en hardware como en software era mejor.

Otros opinan que al tener Keccak solamente una función de compresión, las salidas de bits de longitud largas requieren más número de pequeños bloques de mensaje para ser procesado por cada llamada a la función de compresión y que por tanto Keccak funciona más lento para las variantes de salida de bits de mayor longitud.

También hay partidarios de utilizar el algoritmo de la madeja, Skein, a modo personal.

Muchas personas opinan que SHA-3 no es necesario y que SHA-2 estaba bien, pero que lo importante de estos concursos no es simplemente el resultado sino lo que se despierta en la comunidad criptoanalista en general y las enseñanzas aprendidas acerca de cifradores de flujo.

Hay criptoanalistas que piensan que tanto Blake, debido a su velocidad y la utilización del cifrador Salsa-20, como Skein debido a su diseño de cifrador de bloque merecían ganar. Opinan que además ambos son los más rápidos que Keccak la cual quedaría en tercer lugar. Por otro lado BLAKE Y SKEIN tienen un tamaño más pequeño y así pueden caber mejor en sistemas pequeños.

A parte de detractores o partidarios de la función resultante o del concurso en general, criptoanalistas opinan que los 5 finalistas del concurso eran muy buenos y que cualquiera que hubiera sido la función ganadora hubiera estado bien.

Bruce Schneier

Bruce Schneier desde un principio ha estado conforme con la elección realizada por el NIST argumentando su conformidad con que la función SHA-3 no tuviera nada que ver con la familia de funciones SHA2 puesto que el cambio es bueno. También estaba de acuerdo con el grado de seguridad de la función, además de estarlo ya con las cuatro funciones de la familia SHA-2 antecesoras, eso sí dejando la puerta abierta a un futuro algoritmo.

CAPÍTULO 5

ESTADO ACTUAL DE LA FUNCIÓN GANADORA: FIPS 202

En Agosto de 2013 John Kelsey, trabajador del NIST, anunció algunos cambios en la función Keccak concernientes a la reducción de los niveles de seguridad de la misma con el objetivo de mejorar su rendimiento de software.

El NIST ha hecho pública una nueva convocatoria para recibir análisis y comentarios sobre la función SHA-3 (Keccak). La función ha sido versionada por el FIPS (Federal Information Processing Standard), comité encargado de que se cumplan el conjunto de directrices de seguridad para agencias federales y para empresas que trabajan con ellas.

El objetivo es que mediante SHA-3 se consiga aumentar la seguridad de FIPS180-4, algoritmo utilizado hasta ahora y específico de la funciones de la familia SHA-1 y SHA-2, y dar paso a la nueva generación de la norma FIPS202.

En FIPS202 se especifican seis funciones “esponja” basadas en el algoritmo Keccak. Cuatro de las funciones de hash criptográficas son de longitud fija, y las otras dos funciones son extensibles en su salida. Las cuatro funciones de longitud fija ofrecerán alternativas para SHA-2 y las extensibles pueden ser utilizadas en numerosas aplicaciones.

Las cuatro funciones de longitud fija se denominarán SHA3-224, SHA3-256, SHA3-384 y SHA3-512 y las dos funciones extensibles se denominarán SHAKE128 y SHAKE256.

El NIST dejará el algoritmo y el borrador de FIPS202 abierto a comentarios públicos durante para 90 días, plazo que finaliza el 26 de agosto. Después de ese tiempo el algoritmo se incorporará en la versión final de FIPS202 y será publicado.

Los cambios en la función se han hecho tema de debate y de desconfianza dados los documentos revelados por Edward Snowden en los que se hace entrever una posible intención por parte de la NSA de debilitar intencionadamente estándares criptográficos.

NIST corre el riesgo de publicar un algoritmo en el que nadie confíe y que nadie, excepto los que estén obligados, lleguen a utilizar.

Criptoanalistas de prestigio como Bruce Schneier opinan que se debería publicar el algoritmo tal y como se eligió en un principio, sin los cambios hechos a posteriori. A pesar de esto opinan que los cambios que se han realizado no han sido sugeridos por la NSA ni que estos hagan que la NSA pueda vulnerar el algoritmo de una manera más fácil. También opinan que los cambios realizados por el NIST son de buena fe y que el resultado será una función mejor en cuanto a seguridad y rendimiento.

Para terminar pongo una reflexión de Bruce Schneier sobre los cambios realizados en la función Keccak:

"*Mi problema con los cambios no es criptográfico, es perceptiva. Hay tan poca confianza en la NSA ahora mismo, que la desconfianza se refleja en el NIST. Me preocupa que el algoritmo modificado no sea aceptado por una comunidad de seguridad comprensiblemente escépticos, y que como resultado nadie vaya a utilizar SHA-3.*

Este es un resultado pésimo. NIST ha hecho un gran trabajo con este tipo de competiciones tanto con AES y como ahora con SHA-3. Este es otro efecto de las acciones de la NSA drenando la confianza fuera de Internet. [Ref.91]

CAPÍTULO 6

SEGURIDAD EN LA ACTUALIDAD, CASO SNOWDEN

Edward Joseph Snowden es un ex trabajador de la CIA (Agencia Central de Inteligencia de los Estados Unidos) y de la NSA que en Junio de 2013 hizo públicos varios documentos altamente secretos sobre los programas de vigilancia masiva PRISM y XKeyscore utilizados por Estados Unidos para el espionaje a escala mundial, así como la existencia de tratados secretos y de acuerdos bilaterales para la transferencia masiva de metadatos, registros y otras informaciones.

PRISM y XKeyscore accedían a los servidores de las empresas más importantes de comunicación para captar correos electrónicos, videos, chat de voz, transferencia de archivos, o perfiles de redes sociales, así como contactos, geolocalizaciones, fotografías y aplicaciones o mensajes. El objetivo era crear perfiles.

En concreto, según se ha declarado, la intención era la búsqueda y el análisis de metadatos o información relativa al flujo de llamadas, pero no el contenido de las llamadas en sí.

Los metadatos y su carácter repetitivo podían ser analizados en busca de patrones que cruzados con bases de datos se podían convertir en números de objetivos.

Algunas de las empresas, aplicaciones móviles, sistemas operativos y redes informáticas que se han visto implicadas han sido:

- Empresas: Microsoft, Google, Apple, Yahoo!, Dropbox, Facebook, AOL, Verizon, Vodafone o Global Crossing.
- Aplicaciones móviles: Angry Birds y Google Maps han sufrido la intrusión de software espía.
- Sistemas Operativos: iOS, Android han visto rota de seguridad de sus sistemas y BlackBerry la violación de sus cifrados.
- Redes informáticas: Hotmail, Outlook y Gmail.
- Otros: La intercepción de miles de millones de llamadas y registros telefónicos o el almacenamiento de transacciones financieras electrónicas.

Se ha dicho que estas empresas facilitaban supuestamente la información requerida al Gobierno de Estados Unidos mediante el pago de alta cuantías o voluntariamente. Ante dichas acusaciones, las empresas han declarado cuidar al máximo la privacidad de sus usuarios no proporcionando datos a organismos gubernamentales a no ser mediante orden judicial o mediante peticiones de acuerdo con la ley. Además han declarado el desconocimiento por su parte de que terceros estuvieran accediendo a sus servidores.

Esta vigilancia se hizo en colaboración de países aliados de Estados Unidos, en concreto la UKUSA (United Kingdom-United States Security Agreement), una alianza de naciones de habla inglesa formada en 1946 por Estados Unidos, Reino Unido, Canadá, Australia y Nueva Zelanda con el objetivo inicial de recolectar información y servir al sistema de monitorización de ECHELON.

Las personas afectadas por este espionaje han sido cuantifican en millones de personas alrededor del mundo incluyendo líderes mundiales como jefes de estado e importantes empresarios.

En Estados Unidos no existe ninguna ley en contra de la revelación de información clasificada. A la hora de penalizar revelaciones relativas a defensa nacional se utilizan salvedades legales y solo los tribunales pueden decidir si la información revelada está relacionada con la defensa nacional o no.

Las acciones de Edward Snowden han sido calificadas de "asunto criminal" y se encuentra buscando asilo político mientras continua desaparecido. Presumiblemente se encuentra en Moscú aunque en un principio su paradero se ubicó en Hong Kong.

Este caso ha puesto en entredicho la vulnerabilidad de los derechos fundamentales de la población, en concreto el derecho que tiene toda persona a que se respete y proteja su vida privada y la inviolabilidad de su hogar, poniendo en jaque la efectividad del sistema jurídico respecto a este tema, que no ha podido poner la limitación entre los derechos fundamentales de los ciudadanos frente a la seguridad nacional de los Estados Unidos de América.

Como reflexión, la evolución del ámbito digital, al mismo tiempo que nos ayuda a mantener nuestra privacidad, está permitiendo que terceros la puedan vulnerar de una manera fácil y con poco.

6.1 LOS PROGRAMAS DE VIGILANCIA UTILIZADOS

PRISM: Programa de vigilancia electrónica confidencial utilizado por la NSA de Estados Unidos desde 2007 con el fin de espiar a ciudadanos Estadounidenses que tengan contacto con personas que no residan en el país y también a ciudadanos no residentes en Estados Unidos.

XKeyscore: "Sistema informático secreto utilizado por la NSA de Estados Unidos para la búsqueda y análisis de datos de internet" [*Ref. 84*] que se ejecuta de manera conjunta con otros organismos internacionales como la Dirección de Señales de Defensa de Australia o la Oficina de Seguridad de Comunicaciones del Gobierno de Nueva Zelanda.

XKeyscore es utilizado para analizar el lenguaje utilizado en correos electrónicos interceptados e identificar y espiar a personas de nacionalidad extranjera. En concreto ha sido utilizado en América Latina, específicamente Colombia, Ecuador, México y Venezuela.

6.2 INFORMADORES DE TODOS LOS TIEMPOS

Durante toda la historia Estados Unidos se ha visto amenazado por filtradores que han sacado a la luz numerosos escándalos políticos. Algunos de los informadores más conocidos son:

- Daniel Ellsgerg: desveló los papeles del Pentágono en los que se hacía ver del conocimiento desde un principio por parte de los Estados Unidos de que la guerra de Vietnam posiblemente no podría ser ganada y de las numerosas víctimas que traería consigo.
- William Mark Felt "Garganta Profunda": ayudó a periodistas del periódico Washington Post a descubrir el espionaje telefónico impulsado por el presidente de Estados Unidos Richard Nixon contra el Partido Demócrata además de destapar la obstrucción a la Justicia con el intento de tapar el robo de la oficinas Watergate. Esto provocó la dimisión de Nixon.
- Frederic Whitehurst: denunció malas prácticas en el FBI consistentes en irregularidades en el trabajo de los forenses y en malas prácticas del laboratorio de explosivos con el fin de adulterar pruebas. Gracias a sus revelaciones se produjo una de las reformas más importantes del código interno del FBI.
- Bradley Manning: facilitó archivos secretos al portal WikiLeaks, portal de internet encargado de difundir documentos confidenciales de fuentes anónimas, sobre las guerras de Afganistán y de Irak, entre ellos un video en el que un helicóptero militar norteamericano mata a civiles en Irak, además de cables diplomáticos de las embajadas del mudo.
- Julian Assange: es el caso más actual que tenemos anterior a Edward Snowden. Es el fundador del portal web WikiLeaks y autor de la mayor filtración de cables.

6.3 PROGRAMAS DE ESPIONAJE EN TODOS LOS TIEMPOS

La vigilancia mundial fue un hecho revelado en los años 70 aunque no se hizo conocido hasta el descubrimiento de ECHELON en los años 80 y su posterior confirmación en los años 90.

En los años 70 se hicieron públicas las colaboraciones de Estados Unidos mediante la publicación del acuerdo UKUSA y la existencia del GCHQ (Cuartel General de Comunicaciones del Gobierno de Reino Unido).

En 1988 la red ECHELON fue dada a conocer por Margaret Newsham haciendo públicas el espionaje que la NSA estaba haciendo sobre las llamadas telefónicas de un miembro del Congreso estadounidense.

ECHELON está considerada la mayor red de espionaje y análisis encargada de interceptar comunicaciones de la historia. Controlada por la UKUSA su objetivo era capturar comunicaciones por radio y satélite, así como llamadas de teléfono, faxes y correos electrónico dentro y fuera de las fronteras de Estados Unidos.

Tras los atentados del 11 de septiembre, se recrimino a la NSA por no haber previsto los atentados a pesar del proyecto Traiblazer pensado para analizar datos alojados en redes de comunicaciones.

CAPÍTULO 7
PLANIFICACIÓN Y PRESUPUESTO

En este apartado se detallan aspectos referentes a la gestión del proyecto como son la planificación del trabajo así como un presupuesto mediante el cual se realiza un análisis económico del mismo.

7.1 PLANIFICACIÓN

En este apartado se muestra la planificación inicial diseñada para el desarrollo del proyecto.

Esta planificación se ha intentado seguir estrictamente y el resultado ha sido que la planificación inicial ha coincidido con la planificación final, la desviación ha sido prácticamente inexistente y por ello se ha decidido plasmar es esta memoria solamente la inicial. Cabe destacar que el presupuesto que se mostrará en el siguiente apartado corresponde al presupuesto calculado inicialmente dado este hecho.

CONSIDERACIONES PREVIAS

- Jornadas laborales de 6,5625 horas/día.
- Número de días trabajados al mes 20.
- Fechas de inicio y fin:

- 16 de Mayo de 2011 – 10 de Junio de 2011→ 20 días trabajados.
- 14 de Noviembre de 2011 – 09 de Diciembre de 2011 → 20 días trabajados.
- 16 de Enero de 2012 – 10 de Febrero de 2012 → 20 días trabajados.
- 22 de Octubre de 2012 – 02 de Noviembre de 2012 → 10 días trabajados.
- 27 de Mayo de 2013 – 07 de Junio de 2013 → 10 días trabajados.
- 5 de Junio 2014 – 5 de Julio de 2014 → 20 días trabajados.

Esto hace un total de 100 días trabajados.

En la siguiente tabla se muestra el número de días dedicado a cada parte del proyecto:

Actividad	Fecha de inicio	Fecha de fin	Total de días
Planificación inicial	16/05/2011	16/05/2011	1 día
Estudio del estado del Arte	17/05/2011	18/05/2011	2 días
Estudio del alcance del proyecto	19/05/2011	20/05/2011	2 días
Búsqueda y análisis de información sobre criptografía	23/05/2011	27/05/2011	5 días
	30/05/2011	03/06/2011	5 días
	06/06/2011	10/06/2011	5 días
	14/11/2011	18/11/2011	5 días
Búsqueda y análisis de información sobre SHA-3	21/11/2011	25/11/2011	5 días
	28/11/2011	02/12/2011	5 días
	05/12/2011	09/12/2011	5 días
	16/01/2012	20/01/2012	5 días
	23/01/2012	27/01/2012	5 días
	30/01/2012	03/02/2012	5 días
	06/02/2012	10/02/2012	5 días

Modificaciones entre fases	22/10/2012	26/10/2012	5 días
Búsqueda información desenlace concurso SHA-3	29/10/2012	02/11/2012	5 días
	27/05/2013	31/05/2013	5 días
	03/06/2013	07/06/2013	5 días
Búsqueda información función resultante y caso Snowden	05/06/2014	06/06/2014	2 días
	09/06/2014	13/06/2014	5 días
	16/06/2014	20/06/2014	5 días
Documentación	23/06/2014	27/06/2014	5 días
Modificaciones finales	30/06/2014	02/07/2014	3 días

Tabla 18. Planificación del Proyecto

	Nombre	Duracion	Inicio	Terminado	Predecesores
1	Planificación inicial	1 day?	16/05/11 8:00	16/05/11 17:00	
2	Estudio del estado del Arte	2 days?	17/05/11 8:00	18/05/11 17:00	1
3	Estudio del alcance del proyecto	2 days?	19/05/11 8:00	20/05/11 17:00	2
4	Búsqueda y análisis de información sobre criptografía	15 days?	23/05/11 8:00	10/06/11 17:00	3
5	Búsqueda y análisis de información sobre criptografía	5 days?	14/11/11 8:00	18/11/11 17:00	
6	Búsqueda y análisis de información sobre SHA-3	15 days?	21/11/11 8:00	9/12/11 17:00	5
7	Búsqueda y análisis de información sobre SHA-3	20 days?	16/01/12 8:00	10/02/12 17:00	
8	Búsqueda y análisis de información del desenlace del concurso SHA-3	10 days?	22/10/12 8:00	2/11/12 17:00	
9	Búsqueda y análisis de información del desenlace del concurso SHA-3	10 days?	27/05/13 8:00	7/06/13 17:00	
10	Búsqueda y análisis de información sobre función resultante y caso Snowden	12 days	5/06/14 8:00	20/06/14 17:00	
11	Documentación	5 days?	23/06/14 8:00	27/06/14 17:00	10
12	Modificaciones finales	3 days?	30/06/14 8:00	2/07/14 17:00	11

Figura 49. Tareas del Proyecto

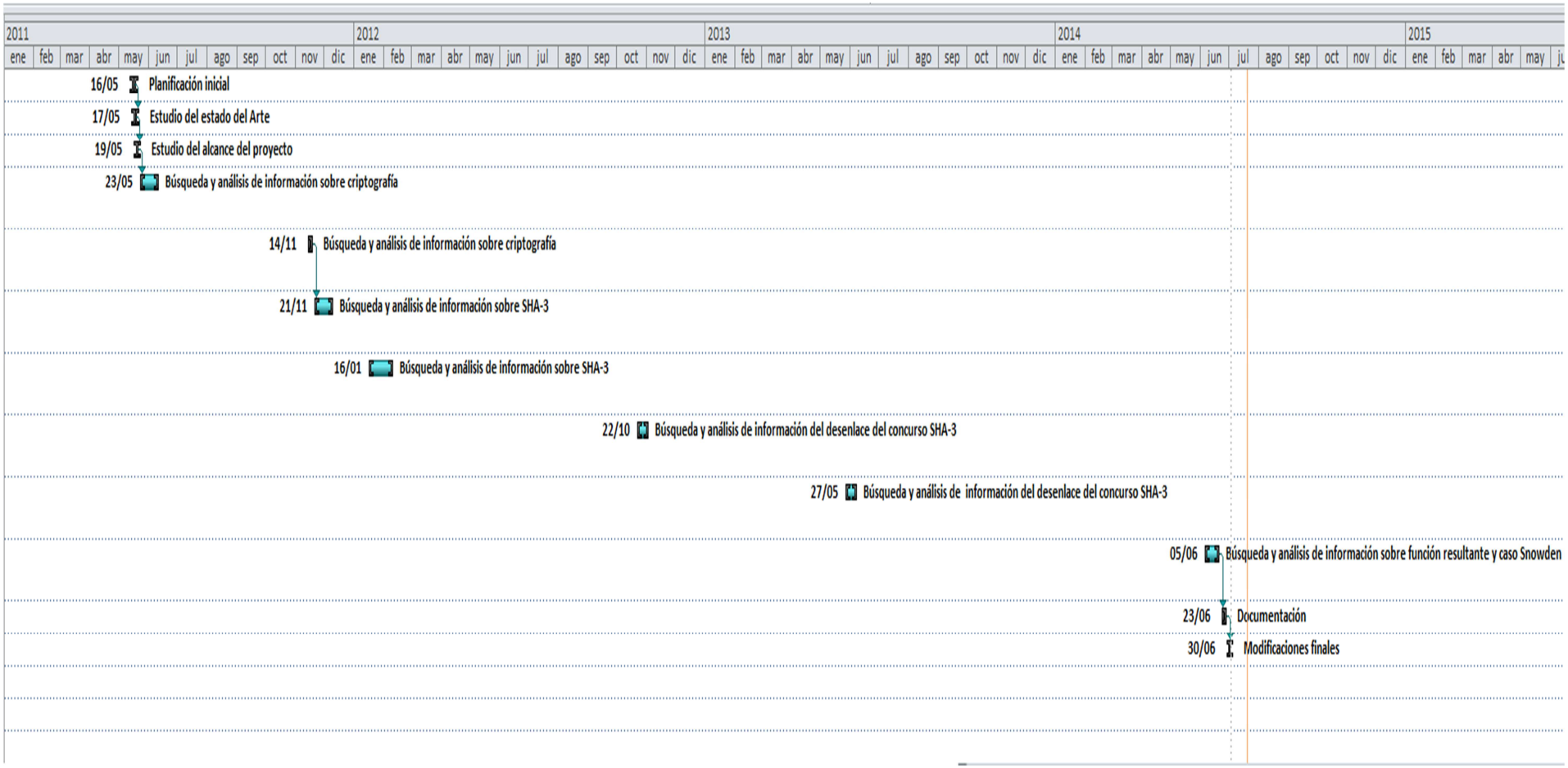


Figura 50. Diagrama de Gantt

7.2 PRESUPUESTO

Se pasa a la realización del presupuesto económico de los elementos utilizados para la ejecución de este proyecto.

Las cantidades calculadas presentan el IVA incluido y el total sin este impuesto aparece al final.

El presente presupuesto se ha realizado atendiendo a diferentes tipos de coste:

- Coste de personal teniendo en cuenta que se trabaja una media de 20 días al mes y que se cobran 50€ la hora descontando impuestos.
- Coste por uso de equipos informáticos.
- Gastos generales, beneficio, riesgos e impuestos (IVA).

1.- Autora: MARIAN MARTÍN SIERRA

2.- Departamento: SEGURIDAD DE LA INFORMACIÓN

3.- Descripción del Proyecto:

- Título CONCURSO NIST. ANÁLISIS DEL CONCURSO (2007-2012)
- Duración (meses): 5

4.- Presupuesto total del Proyecto (valores en Euros):

38136,9 Euros (IVA incluido)

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Marian Martín Sierra	03910043C	Ingeniero Técnico en informática de Gestión. Graduado en Informática	1 Hombre	6562,5	32812,5	
			Hombres mes 1	**Total**	32812,5	(IVA incluido)

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)

Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d] (Euro)
Ordenador portátil Asus AS3S	600,00	100	5	60	50,00
				Total	50,00 (IVA incluido)

[d] Fórmula de cálculo de la Amortización:

$\frac{A}{B} xCxD$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

OTROS COSTES DIRECTOS DEL PROYECTO [e]

Descripción	Empresa	Costes imputable (Euro)
Programas informáticos		120,00
Luz y calefacción		120,00
Otros materiales		60,00
	Total	300,00 (IVA incluido)

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	Presupuesto Costes Totales (Euro)
Personal	32812,5
Amortización	50,00
Costes Directos	300,00
Riesgos (5%)	1658,125
Beneficio (10%)	3316,25
Total	**38136,9**
Total sin IVA (21%)	**30128,15**

El presupuesto total de este proyecto asciende a la cantidad de 38136,9 Euros.

Leganés a 18 de Julio de 2014

La ingeniera proyectista

Fdo. Marian Martín Sierra

CAPÍTULO 8

GLOSARIO

AES	Advanced Encryption Standard
CBC-MAC	Cipher Block Chaining Message Authentication Code
CRHF	Colision Resistant Hash Function
DES	Data Encryption Standard
FIPS	Federal Information Processing Standard
FPGA	Field Programmable Gate Array
HMAC	Keyed-Hash Message Authentication Code
MASH-1	Modular Arithmetic Secure Hash
MAC	Message Authenticacion Codes
MDC	Modification Detection Codes
MD2	Message Digest 2
MD4	Message Digest 4
MD5	Message Digest 5
NIST	Instituto nacional de estándares y tecnología

NSA	Agencia de seguridad nacional
OWHF	One Way Hash Function
PDA	Punto de acceso a internet
SHA	Secure Hash Algorithm
SHA-1	Secure Hash Algorithm 1
SHA-2	Secure Hash Algorithm 2
SHA-3	Secure Hash Algorithm 3
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UNIX	Sistema Operativo Portable, Multitarea y Multiusuario
UMAC	Message Authentication Code Based on Universal Hashing

CAPÍTULO 9

CONCLUSIONES

En este trabajo se ha realizado un estudio de concurso SHA-3, concurso promovido por el NIST con el objetivo de encontrar un nuevo algoritmo que sustituya al algoritmo de hash SHA-1 utilizado hasta el momento a escala mundial como estándar de cifrado.

Para ello ha sido necesario un estudio previo sobre criptografía en general así como de algunos de los algoritmos criptográficos de función resumen más utilizados hasta la fecha.

Este trabajo ha pasado por varias fases en las que el concurso ha pasado por sus distintas fases hasta llegar a su finalización y dando como resultado la elección del algoritmo Keccak como ganador.

Durante estos años el trabajo ha sufrido varios parones por circunstancias familiares y laborales y se ha tenido que ir modificando el proyecto añadiendo la nueva información de la que se disponía.

Además del análisis del concurso, de sus candidatos y de sus fases, y a parte del estudio sobre la criptografía en general, se ha hecho un breve resumen sobre el caso Snowden y sobre los problemas de seguridad que Estados Unidos está sufriendo y ha sufrido.

Gracias a este proyecto se ha podido hacer un documento de cabecera para que toda persona interesada en el concurso SHA-3 y en la criptografía en general pueda de una manera sencilla estar al día de nuevos avances en esta materia.

Se han analizado los 5 finalistas de manera que quedan evidentes sus diferencias en cuanto a diseño y en cuanto a rendimiento se refiere. Además ha quedado plasmada la visión de la comunidad criptoanalista sobre el concurso en general y sobre la función ganadora en concreto de manera subjetiva y objetiva.

A pesar de toda la controversia, tras todos estos años este concurso ha servido además de para crear un nuevo algoritmo criptográfico de función resumen utilizado a escala mundial, para algo mucho más importante que es despertar en la población el interés por la criptografía independientemente del resultado.

9.1 TRABAJO FUTURO

Dado el periodo para análisis y estudio ofrecido por el NIST para el análisis de Keccak y su futura incorporación al FIPS202 que finaliza el 26 de agosto, una posible ampliación es la búsqueda de resultados por parte de la comunidad criptoanalista y su posible análisis.

Estos estudios todavía no han sido publicados, buscando por Internet no se han encontrado publicaciones por parte de la comunidad acerca de los cambios realizados por el NIST para que la función concuerde con las especificaciones técnicas requeridas.

Tras este periodo de 90 días, cuando los criptógrafos hagan públicos estos datos, se podrá hacer una comparativa entre ellos, para ver la objetividad y veracidad de los mismos.

Por otro lado, cuando transcurra este periodo, NIST hará pública FIPS202. Al igual que se hará con los estudios de criptoanalistas, FIPS202 también podrá ser analizada en cuanto a seguridad y rendimiento se refiere. También se podrán analizar estudios independientes que aportarán mayor claridad al entendimiento de la función.

CAPÍTULO 10

REFERENCIAS

[Ref.01] Noticia digital sobre los ataques a SHA-1. Disponible [Internet]: <http://www.wikilearning.com/articulo/picadillo_digital-picadillo_digital/3364-1> (Noticia extraída de: <http://www.kriptopolis.org> 07 Agosto 2005). Accedido en Mayo 2011.

[Ref.02] El algoritmo más seguro del mundo es buscado. Disponible [Internet]: <http://tenoch.scimexico.com/2010/12/22/el-algoritmo-mas-seguro-del-mundo-es-buscado/>. Accedido en Mayo 2011.

[Ref.03] Matthew Fanto. NOTACON 2: Recent Attacks Against Hash Functions. Disponible [Internet]: <http://www.youtube.com/watch?v=b087ZRzZJbE>. Accedido en Noviembre 2011.

[Ref.04] Función Hash. Disponible [Internet]: <http://es.wikipedia.org/wiki/Funci%C3%B3n_hash>. Accedido en Junio 2011.

[Ref.05] Función Hash criptográfica. Disponible [Internet]: <http://es.wikipedia.org/wiki/Funci%C3%B3n_hash_criptogr%C3%A1fica>. Accedido en Junio 2011.

[Ref.06] Cryptographic hash. Disponible [Internet]: <http://en.wikipedia.org/wiki/Cryptographic_hash_function>. Accedido en Junio 2011

[Ref.07] Microsoft Research, Silicon Valley Campus. Ilya Mironov. Hash functions: Theory, attacks, and applications. 14 de Noviembre de 2005. Disponible [Internet]: <http://research.microsoft.com/pubs/64588/hash_survey.pdf>. Accedido en Noviembre 2011.

[Ref.08] SHA-1. Disponible [Internet]: <http://en.wikipedia.org/wiki/SHA-1>. Accedido en Noviembre 2011.

[Ref.09] Código de detección de manipulaciones. Disponible [Internet]: <http://es.wikipedia.org/wiki/C%C3%B3digo_de_detecci%C3%B3n_de_modificaciones>. Accedido en Febrero 2012.

[Ref.10] Departamento de Seguridad en Redes de Comunicaciones. María Dolores Cano Baños. Bloque II Sistemas de autenticación: Algoritmos generadores de autenticadores. Disponible [Internet]: <http://ocw.bib.upct.es/pluginfile.php/6706/mod_resource/content/1/autenticadores.swf>. Accedido en Diciembre 2011.

[Ref.11] Funciones Hash. 22 de Diciembre de 2005. Disponible [Internet]: <http://foro.elhacker.net/criptografia/funciones_de_hash-t100025.0.html>. Accedido en Junio 2011.

[Ref.12] Funciones hash en criptografía. Disponible [Internet]: <http://criptosec.unizar.es/doc/tema_c7_criptosec.pdf> Accedido en Junio 2011.

[Ref.13] Universidad Politécnica de Madrid. Jorge Ramió Aguirre. Funciones Hash en Criptografía. 1 de Marzo de 2005. Disponible [Internet]: <http://es.scribd.com/doc/4680022/Seguridad-Informatica-y-Criptografia-Funciones-Hash-en-Criptografia>. Accedido en Junio 2011.

[Ref.14] Algoritmos de HASH. Disponible [Internet]: <http://es.scribd.com/doc/46651/Algoritmos-de-HASH> (Articulo creado para postear en <http://gaussianos.com>). Accedido en Junio 2011.

[Ref.15] Maytée Odette López Catalá, Virgilio Zuaznabar Mazorra. 2008. Monografía Funciones Hash Criptografía. Disponible [Internet]: <http://www.plusformacion.com/Recursos/r/Monogafia-Funciones-HASH-Criptografia>. Accedido en Junio 2011.

[Ref.16] Gerald Hines. SHA-1 Hash Tutorial. Disponible [Internet]: <http://www.youtube.com/watch?v=aLvwpJcOy6s>. Accedido en Noviembre 2011.

[Ref.17] Ataque de cumpleaños. Disponible [Internet]: <http://es.wikipedia.org/wiki/Ataque_de_cumplea%C3%B1os>. Accedido en Noviembre 2011.

[Ref.18] IEFD EP 10 - Hacking Basics – MD5. Disponible [Internet]: <http://www.youtube.com/watch?v=96fAX0beduE>. Accedido en Noviembre 2011.

[Ref.19] Introducción a la criptografía moderna con Python (3): MD5. 17 Noviembre 2011. Disponible [Internet]: <http://blog.hackxcrack.es/2011/11/introduccion-la-criptografia-moderna_17.html>. Accedido en Enero 2012.

[Ref.20] Universidad Politécnia de Pachuac. MD5. Disponible [Internet]: <http://maytics.web44.net/web_documents/md5.pdf>. Accedido en Enero 2012.

[Ref.21] MIT Laboratory for Computer Science and RSA Data Security R. Rivest. El Algoritmo de Resumen de Mensajes MD5. Abril 1992. Traducción por Rubén Alfonso Francos. Octubre 2003. Disponible [Internet]: <http://www.rfc-es.org/rfc/rfc1321-es.txt>. Accedido en Enero 2012.

[Ref.22] Departamento de Informática de la Universidad Politécnica de Madrid. Software instalable gratuito. CriptoRES: hash MD5 y SHA-1. Disponible [Internet]: <http://www.criptored.upm.es/software/sw_m001h.htm>. Accedido en Enero 2012.

[Ref.23] Algoritmos HASH (II): Atacando MD5 Y SHA-1. Disponible [Internet]: <http://gaussianos.com/algoritmos-hash-ii-atacando-md5-y-sha-1/>. Accedido en Noviembre de 2012.

[Ref.24] Merkle-Damgard construction. Disponible [Internet]: <http://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd_construction>. Accedido en Febrero 2012.

[REF.25] A Framework for Iterative Hash Functions — HAIFA. Disponible [Internet]: <http://eprint.iacr.org/2007/278.pdf>. Accedido en Febrero 2012.

[Ref.26] Daniel J. Bernstein. Departamento de Matemátias y Ciencias de la Universidad de Illiois en Chicago. ChaCha, una variante de Salsa20. Disponible [Internet]: <http://cr.yp.to/chacha/chacha-20080128.pdf>. Accedido en Febrero 2012.

[Ref.27] Brian Baldwin, Hanley Neil Hamilton Marcos, Liang Lu, Andrew Byrne, O'Neill Maire y William P. Marnane. FPGA implementaciones de la Segunda Ronda de los candidatos de SHA-3. Disponible [Internet]: <http://csrc.nist.gov/groups/ST/hash/sha3/Round2/Aug2010/documents/papers/BALDWIN_FPGA_SHA3.pdf>. Accedido en Mayo 2011.

[Ref.28] Abdulkadir Aysu Akin, Aydin, Onur Ulusel y Savas Erkay. Implementaciones eficientes de hardware de alto rendimiento de los candidatos de SHA-3. Disponible [Internet]: <http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/SAVAS_SHA3_NIST_final.pdf>. Accedido en Mayo 2011.

[Ref.29] Guo Xu, Huang Sinan, Leyla Nazhandali y Schaumont Patrick. Feria y Evaluación del Desempeño Integral de los candidatos 14 de la Segunda Ronda SHA-3, Implementaciones ASIC. Disponible [Internet]: <http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/SCHAUMONT_SHA3.pdf>. Accedido en Mayo 2011.

[Ref.30] SHA-3. Disponible [Internet]: <http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations>. Accedido en Mayo 2011.

[Ref.31] Victor Williamson, Trevor Rundell, Oliver Yeh. Analysis of SHA-3 Hash Functions. Disponible [Internet]: <http://courses.csail.mit.edu/6.857/2009/sha3/group7.pdf>. Accedido en Mayo 2011.

[Ref.32] Bart Preneel. The cryptographic hash function crisis and the SHA-3 competition. Disponible [Internet]: <http://www.forth.gr/onassis/lectures/2010-06-28/presentations/The_cryptographic_hash_function_crisis_and_the_SHA_3_competition.pdf>. Accedido en Mayo 2011.

[Ref.33] <http://staff.aist.go.jp/akashi.satoh/SASEBO/en/sha3/others.html>. Accedido en Mayo 2011.

[Ref.34] SHA-3 proposal BLAKE. Disponible [Internet]: <http://131002.net/blake/>. Accedido Mayo 2011.

[Ref.35] Aumasson Jean-Philippe, Henzen Luca, Willi Meier, y Rafael C.-W. Phan. Propuesta BLAKE para SHA-3 (versión 1.3). Disponible [Internet]: <http://131002.net/blake/blake.pdf>. Accedido en Mayo 2011.

[Ref.36] Luca Henzen, Aumasson Jean-Philippe, Willi Meier, y Rafael C.-W. Phan. VLSI Caracterización de la función hash criptográfica BLAKE. IEEE T VLSI, 2010.

Disponible [Internet]: <http://131002.net/data/papers/HAMP10.pdf>. Accedido en Mayo 2011.

[Ref.37] Blake. Disponible [Internet]: <http://ehash.iaik.tugraz.at/wiki/BLAKE>. Accedido en Mayo 2011.

[Ref.38] Algoritmo BLAKE. Disponible [Internet]: <http://en.wikipedia.org/wiki/BLAKE_(hash_function)>. Accedido en Mayo 2011.

[Ref.39] Jean-Luc Beuchat, Eiji Okamoto, y Teppei Yamazaki. Pacto implementaciones de BLAKE-32 y Blake-64 en FPGA. CAII informe 2010/173. Disponible [Internet]: <http://eprint.iacr.org/2010/173.pdf>. Accedido en Mayo 2011.

[Ref.40] Jean-Philippe Aumasson. The Cryptographic Hash Function BLAKE. 21 de Septiembre de 2011. Disponible [Internet]: <https://www.youtube.com/watch?v=PgpJNRnx6eY>. Accedido Enero 2012.

[Ref.41] GROSTL. Disponible [Internet]: <http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/Feb2009/documents/Groestl_Presentation_SHA-3_NIST_Conf_2page_version.pdf>. Accedido en Mayo 2011.

[Ref.42] Gauravaram Praveen, R. Lars Knudsen, Matusievicz Krystian, Mendel Florian, Rechberger cristiana, Schläffer Martin, y Søren S. Thomsen. Grøstl - - un candidato de SHA-3. 02 de Marzo de 201. Disponible [Internet]: <http://www.groestl.info/Groestl.pdf>. Accedido en Mayo 2011.

[Ref.43] <http://www.groestl.info/>. Accedido en Mayo 2011.

[Ref.44] Groestl. Disponible [Internet]: <http://ehash.iaik.tugraz.at/wiki/Groestl>. Accedido en Mayo 2011.

[Ref.45] Bernhard Jungk y Reith Steffen. Las On FPGA-based implementations of Grøstl. CAII informe 2010/260. Disponible [Internet]: <http://eprint.iacr.org/2010/260.pdf>. Accedido en Mayo 2011.

[Ref.46] Hash Function JH. Disponible [Internet]: <http://www3.ntu.edu.sg/home/wuhj/research/jh/>. Accedido en Mayo 2011.

[Ref.47] JH. Disponible [Internet]: <http://ehash.iaik.tugraz.at/wiki/JH>. Accedido en Mayo 2011.

[Ref.48] Florian Mendel y Soren S. Thomsen. An Observation on JH-512. Disponible [Internet]: <http://ehash.iaik.tugraz.at/uploads/d/da/Jh_preimage.pdf>. Accedido en Febrero 2012.

[Ref.49] Hongjun Wu. September 15, 2009. The Hash Function JH. Disponible [Internet]: <http://ehash.iaik.tugraz.at/uploads/1/1d/Jh20090915.pdf>. Accedido en Mayo 2011.

[Ref.50] Hongjun Wu. 16 Enero de 2011. The Hash Funciton JH. Disponible [Internet]: <http://www.aceparadis.horizon-host.com/pubs/jh20110116.pdf>. Accedido en Mayo 2011.

[Ref.51] Rishiraj Bhattacharyya1, Avradip Mandal, y Mridul Nandi. Indian Statistical Institute, Kolkata, India, Université du Luxembourg, Luxembourg y NIST, USA and Computer Science Departmente, The George Washington University. Security Analysis of the Mode of JH Hash Function. Disponible [Internet]: <http://www.isical.ac.in/~rishi_r/FSE2010-146.pdf>. Accedido en Febrero 2012.

[Ref.52] Jooyoung Lee and Deukjo Hong. Collision Resistance of the JH Hash Function. Disponible [Internet]: <http://eprint.iacr.org/2011/019.pdf>. Accedido en Febrero 2012.

[Ref.53] Keccak. Disponible [Internet]: <http://ehash.iaik.tugraz.at/wiki/Keccak>. Accedido en Mayo 2011.

[Ref.54] Guido Bertoni, Joan Daemen, Michaël Peeters y Gilles Van Assche. Página Web sobre el algoritmo Keccak. The Keccak sponge function family. Disponible [Internet]: <http://keccak.noekeon.org>. Accedido en Mayo 2011.

[Ref.55] Guido Bertoni, Joan Daemen, Michaël Peeters, y Gilles van Assche. KECCAK, función principal de la familia del documento (versión 1.2). 23 de abril de 2009. Disponible [Internet]: <http://keccak.noekeon.org/Keccak-main-1.2.pdf>. Accedido en Mayo 2011.

Ref.56] Strömbergson Joachim. Aplicación de la función hash Keccak en dispositivos FPGA. Disponible [Internet]: <http://www.strombergson.com/files/Keccak_in_FPGAs.pdf>. Accedido en Mayo 2011.

[Ref.57] Documentos presentados en la primera y segunda ronda del concurso en formato pdf sobre el algoritmo Skein. Disponible [Internet]: <http://ehash.iaik.tugraz.at/wiki/Skein>. Accedido en Mayo 2011.

[Ref.58] Página Web de Bruce Schneier <http://www.schneier.com/>. Accedido en Mayo 2011.

[Ref.59] Tillich Stefan. Hardware Aplicación de la madeja, SHA-3 candidatos. CAII informe 2009/159. Disponible [Internet]: <http://eprint.iacr.org/2009/159.pdf>. Accedido en Febrero 2012.

[Ref.60] Jesse Walker, Jeque Farhana, Sanu K. Mathew, y Krishnamurthy Ram. Una Aplicación de hardware de madeja-512. Disponible [Internet]: <http://csrc.nist.gov/groups/ST/hash/sha-

3/Round2/Aug2010/documents/papers/WALKER_skein-intel-hwd.pdf>. Accedido en Mayo 2011.

[Ref.61] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas and Jesse Walker. The Skein Hash Function, First Hash Function Candidate Conference Leuven. Belgica, 26 February 2009. Disponible [Internet]: <http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/Feb2009/documents/Skein%20Presentation%20(Feb%2009).pdf>. Accedido en Mayo 2011.

[Ref.62] Los hombres de largo. La aplicación de la función hash madeja en la plataforma FPGA de Xilinx Virtex-5 (versión 0.7). 2 de febrero de 2009. Disponible [Internet]: <http://www.skein-hash.info/sites/default/files/skein_fpga.pdf>. Accedido en Mayo 2011.

[Ref.63] Olivier Benoit y Thomas Peyrin. Side-Channel Analysis of Six SHA-3 Candidates. Disponible [Internet]: <http://eprint.iacr.org/2010/447.pdf>. Accedido en Mayo 2011.

[Ref.64] George Mason University. Ekawat Homsirikamol, Marcin Rogawski, Kris Gaj George. 21 Diciembre 2010. Comparación del rendimiento sobre el hardware de catorce candidatos en la segunda ronda de SHA-3 usando FPGAs. Disponible [Internet]: <http://eprint.iacr.org/2010/445.pdf>. Accedido en Mayo 2011.

[Ref.65] Bernhard Jungk, Hochschule RheinMain University of Applied Sciences Wiesbaden Russelsheim Geisenheim. Compact implementations of Groestl, JH and Skein for FPGAs. Disponible [Internet]: <http://www.ecrypt.eu.org/hash2011/proceedings/hash2011_09.pdf>. Accedido en Mayo 2011.

[Ref.66] Elena Andreeva, Bart Mennink, Bart Preneel and Marjan Skrobot Dept. Electrical Engineering, ESAT/COSIC y IBBT Katholieke, Universiteit Leuven, Belgium. Security Analysis and Comparison of the SHA-3 Finalists Blake, Groestl, JH, Keccak and Skein. Disponible [Internet]: <http://www.cosic.esat.kuleuven.be/publications/article-2173.pdf>. Accedido en Mayo 2011.

[Ref.67] Kazuyuki Kobayashi, Ikegami Junio, Shin'ichiro Matsuo, Kazuo Sakiyama, y Kazuo Ohta. Evaluación de las Prestaciones de hardware para los candidatos de SHA-3 Uso Sasebo-GII. CAII Eprint informe 2010/010 Disponible [Internet]: <http://eprint.iacr.org/2010/010.pdf>. Accedido en Mayo 2011.

[Ref.68] Ota Shin'ichiro Matsuo, Miroslav Knezevic, Schaumont Patrick, Verbauwhede Ingrid, Akashi Satoh, Kazuo Sakiyama, y Kazuo. ¿Cómo podemos evaluar justa y coherentemente el hardware de los candidatos de SHA-3? Disponible [Internet]: <http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/MATSUO_SHA-3_Criteria_Hardware_revised.pdf>. Accedido en Mayo 2011.

[Ref.69] Tillich Stefan, Feldhofer Martín, Mario Kirschbaum, Plos Thomas Schmidt Jörn-Marc y Alejandro Szekely. De alta velocidad implementaciones de hardware de BLAKE, Grøstl, JH, Keccak y la madeja. CAII Eprint informe 2009/510. Disponible [Internet]: <http://eprint.iacr.org/2009/510.pdf>. Accedido en Mayo 2011.

[Ref.70] Talk: The SHA-3 Zoo. Disponible [Internet]: <http://ehash.iaik.tugraz.at/wiki/Talk:The_SHA-3_Zoo>. Accedido en Mayo 2011.

[Ref.71] Stefan Tillich, Feldhofer Martin, Wolfgang Issovits, Thomas Kern, Kureck Hermann, Mühlberghuber Michael, Georg Neubauer, Reiter Andreas Kofler Armin y Mathias Mayrhofer. Las implementaciones de hardware compacto de los candidatos de

SHA-3 candidatos Blake, Grøstl, y la madeja. CAII informe 2009/349. Disponible [Internet]: <http://eprint.iacr.org/2009/349.pdf>. Accedido en Mayo 2011.

[Ref.72] Brian Baldwin, Andrew Byrne, Hamilton Mark Hanley Neil, Robert P. McEvoy, Pan Weibo, y William P. Marnane. Implementaciones FPGA de los candidatos de SHA-3: Grøstl. IACR Eprint report 2009/342. Disponible [Internet]: <http://eprint.iacr.org/2009/342.pdf>. Accedido en Mayo 2011.

[Ref.73] Bernhard Jungk, Reith Steffen, y Apfelbeck Jürgen. Implementaciones optimizadas FPGA de Grøstl, candidatos de SHA-3. Disponible [Internet]: <http://eprint.iacr.org/2009/206.pdf>. Accedido en Mayo 2011.

[Ref.74] Luca Henzen, Gendotti Pietro, Patrice Guillet, Enrico Pargaetzi, Zoller Martin y Frank K. Gürkaynak. Desarrollo de un Método de Evaluación de hardware para los candidatos de SHA-3. 12 Taller Internacional sobre hardware criptográfico y sistemas incorporados (CHES), 2010. Disponible [Internet]: <http://www.springerlink.com/content/g0115v3272156r06/>. Accedido en Mayo 2011.

[Ref.75] Información sobre Edward Joseph Snowden. Disponible [Internet]: <http://es.wikipedia.org/wiki/Edward_Snowden>. Accedido en Junio 2014.

[Ref.76] Información sobre Edward Joseph Snowden. Disponible [Internet]: <http://www.la-razon.com/suplementos/informe/SNOWDEN-hacker-apocalipsis_0_1876612445.html>. Accedido en Junio 2014.

[Ref.77] Visión general del caso Snowden. Disponible [Internet]: <http://alponiente.com/resumen-del-caso-snowden/>. Accedido en Junio 2014.

[Ref.78] Información sobre el programa de vigilancia PRISM. Disponible [Internet]: <http://es.wikipedia.org/wiki/PRISM_(programa_de_vigilancia)> //

<http://en.wikipedia.org/wiki/PRISM_(surveillance_program)>. Accedido en Junio 2014.

[Ref.79] Información sobre el programa de vigilancia PRISM. Disponible [Internet]: <http://alt1040.com/2013/06/que-es-prism-claves>. Accedido en Junio 2014.

[Ref.80] Información sobre el programa de vigilancia XKeyscore. Disponible [Internet]: <http://es.wikipedia.org/wiki/XKeyscore>. Accedido en Junio 2014.

[Ref.81] Información sobre la vigilancia mundial. Disponible [Internet]: <http://es.wikipedia.org/wiki/Datos_acerca_de_la_vigilancia_mundial_(2013_a_la_fecha)>. Accedido en Junio 2014.

[Ref.82] Información sobre la vigilancia mundial antes de 2013. Disponible [Internet]: <http://es.wikipedia.org/wiki/Datos_acerca_de_la_vigilancia_mundial_(1970-2013)>. Accedido en Junio 2014.

[Ref.83] Información sobre ECHELON. Disponible [Internet]: <http://es.wikipedia.org/wiki/ECHELON>. Accedido en Junio 2014.

[Ref.84] Comentarios sobre SHA-3. Disponible [Internet]: <http://threatpost.com/nist-seeks-public-comment-on-sha-3-crypto-algorithm/106444>. Accedido en Junio 2014.

[Ref.85] Guido Bertoni, Joan Daemen, Michaël Peeters y Gilles Van Assche. Actualizaciones sobre SHA-3. Disponible [Internet]: <http://keccak.noekeon.org/>. Accedido en Junio 2014.

[Ref.86] Bruce Schneier. Comentarios sobre SHA-3. Disponible [Internet]: <https://www.schneier.com/blog/archives/2012/10/keccak_is_sha-3.html>. Accedido en Junio 2014.

[Ref.87] Análisis sobre Keccak. Disponible [Internet]: <http://www.researchgate.net/publication/261295082_Differential_Cryptanalysis_of_Keccak_Variants>. Accedido en Junio 2014.

[Ref.88] Información sobre FIPS202 Disponible [Internet]: <http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf>. Accedido en Junio 2014.

[Ref.89] Noticia en la que se solicitan comentarios públicos y análisis del algoritmo Keccak para su futura implantación en el FIPS202. Disponible [Internet]: <http://www.nist.gov/itl/csd/sha-3-_standard.cfm>. Accedido en Junio 2014.

[Ref.90] Noticia en la que se solicitan comentarios públicos y análisis del algoritmo Keccak para su futura implantación en el FIPS202. Disponible [Internet]: <https://www.federalregister.gov/articles/2014/05/28/2014-12336/announcing-draft-federal-information-processing-standard-fips-202-sha-3-standard-permutation-based>. Accedido en Junio 2014.

[Ref.91] Entrada en el blog de Bruce Schneier. Disponible [Internet]: <https://www.schneier.com/blog/archives/2013/10/will_keccak_sha-3.html>. Accedido en Junio 2014.

[Ref.92] Guido Bertoni, Joan Daemen, Michaël Peeters y Gilles Van Assche. Función Criptográfica RadioGatun. Disponible [Internet]: http://radiogatun.noekeon.org/. Accedido en Mayo 2011.

[Ref.93] The sponge function. Disponible [Internet]: <http://en.wikipedia.org/wiki/Sponge_function>. Accedido en Mayo 2011.

[Ref. 94] Distintos archivos de temas variados en formato pdf disponibles en este portal Web. Disponible [Internet]: <https://eprint.iacr.org/>. Accedido desde Mayo 2011.

CAPÍTULO 11
ANEXO: COMPARATIVAS DE LOS 5 MODELOS ELEGIDOS

En este anexo se muestran los resultados obtenidos al realizar las pruebas de la segunda ronda, donde ya solo había 14 candidatos.

El algoritmo BLAKE aparece con su nombre antiguo, es decir, BLAKE-28, BLAKE-32, BLAKE-48, y BLAKE-64, los cuales fueron renombrados en diciembre de 2010 por BLAKE-224, BLAKE-256, BLAKE-384, y BLAKE-512.

Este anexo se ha realizado durante toda la realización de este proyecto fin de carrera, recopilando datos de cada uno de los documentos consultados, incluyendo numerosos documentos del portal web *http://eprint.iacr.org*.

Hay que tener en cuenta que a la hora de tomar los resultados, para la implementación de FPGA, hay comparar las implementaciones en el mismo dispositivo destino, o al menos en los dispositivos de la misma familia de FPGA.

El tamaño y la funcionalidad de cortes varían entre las familias FPGA, una comparación directa del número de rebanadas en implementaciones en diferentes familias FPGA es problemática

Hay que observar que para facilitar la comparación de los módulos de hardware con ámbitos de aplicación diferentes, éstos están clasificados en tres categorías:

- Totalmente autónomo: el mensaje de entrada se pueden cargar a trozos en el módulo de hardware y ofrece el resumen del mensaje como salida. Todos los cálculos hash ocurren exclusivamente en el módulo de hardware.
- Uso de la memoria externa: en estas implementaciones se utiliza memoria externa para mantener los valores intermedios durante el hash de un mensaje. Estas implementaciones pueden cargar el mensaje de entrada ya sea a través de una interfaz dedicada (similar a una aplicación totalmente autónoma) o desde la memoria externa. Con el fin de alcanzar el rendimiento máximo del módulo de hardware, la memoria externa debe ser lo suficientemente rápida.
- Funcionalidad básica: Este tipo de implementaciones sólo representan una parte importante de la función hash (por ejemplo, la función de compresión), que normalmente permite obtener una estimación de primer orden de las cifras de rendimiento de las implementaciones.

CARACTERÍSTICAS COMPARADAS

- Ámbito de aplicación (descrito anteriormente).
- Detalles de la implementación.
- Tecnología utilizada.
- Número de rebanadas.
- Rendimiento medido Mbits/s.
- Frecuencia de reloj medida en MHz.

Nombre de la función hash	Referencia / HDL	Impl. Ámbito de aplicación	Impl. Detalles	Tecnología	Tamaño	Rendimiento	Frecuencia de reloj
BLAKE-32	Portal Web http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 8 unidades de la función G	Xilinx Virtex-II Pro	3091 rebanadas	1724 Mbit / s	37.0 MHz
BLAKE-32	Portal Web http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 8 unidades de la función G	Xilinx Virtex 4	3087 rebanadas	2235 Mbit / s	48.0 MHz
BLAKE-32	Portal Web http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 8 unidades de la función G	Xilinx Virtex 5	1694 rebanadas	3103 Mbit / s	67.0 MHz
BLAKE-32	Portal Web http://www.vlsi.uwaterloo.ca	Funcionalidad básica	Compresión de la función con 8 unidades de la función G e I / O registros	Altera Stratix III	5435 ALUTs	2.186,2 Mbit / s	46,97 MHz
BLAKE-32	http://eprint.iacr.org/2010/010.pdf	Totalmente autónomo		Xilinx Virtex 5	1660 rebanadas	2676 Mbit / s	115 MHz
BLAKE-32	http://www.ucc.ie/en/crypto/SHA-3Hardware/	Totalmente autónomo		Xilinx Virtex 5	1118 rebanadas	1169 Mbit / s	118,06 MHz
BLAKE-32	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	1660 rebanadas	2676 Mbit / s	115 MHz

BLAKE-64	Portal Web http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 8 unidades de la función G	Xilinx Virtex-II Pro	11.122 rebanadas	1177 Mbit / s	17.0 MHz
BLAKE-64	Portal Web http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 8 unidades de la función G	Xilinx Virtex 4	11.483 rebanadas	1707 Mbit / s	25.0 MHz
BLAKE-64	Portal Web http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 8 unidades de la función G	Xilinx Virtex 5	4329 rebanadas	2389 Mbit / s	35.0 MHz
BLAKE-64	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	1718 rebanadas	1299 Mbit / s	90,91 MHz
Grøstl-224/256	página web http://eprint.iacr.org/2009/206.pdf	Totalmente autónomo	P & Q permutación en paralelo	Xilinx Spartan 3	6136 rebanadas	4520 Mbit / s	88,3 MHz
Grøstl-224/256	página web http://eprint.iacr.org/2009/206.pdf	Totalmente autónomo	P & Q permutación en paralelo	Xilinx Virtex 5	1722 rebanadas	10 276 Mbit / s	200.7 MHz
Grøstl-224/256	documento http://eprint.iacr.org	Funcionalidad básica	P & Q permutación de forma paralela, S-caja en BRAM	Xilinx Spartan 3	4827 rebanadas	3660 Mbit / s	71,53 MHz
Grøstl-224/256	documento http://eprint.iacr.org	Funcionalidad básica	P & Q permutación de forma paralela, S-caja en BRAM	Xilinx Virtex 5	4516 rebanadas	7310 Mbit / s	142,87 MHz
Grøstl-256	http://eprint.iacr.org/2010/010.pdf	Totalmente autónomo		Xilinx Virtex	4057	5171 Mbit / s	101 MHz

				5	rebanadas		
Grøstl-256	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	2391 rebanadas	3242 Mbit / s	101,32 MHz
Grøstl-256	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	2616 rebanadas	7885 Mbit / s	154 MHz
Grøstl-384/512	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo	P & Q permutación en paralelo	Xilinx Spartan 3	20.233 rebanadas	5901 Mbit / s	80.7 MHz
Grøstl-384/512	documento http://eprint.iacr.org	Funcionalidad básica	P & Q paralelo permutación, S-caja en LUT	Xilinx Spartan 3	17.452 rebanadas	3180 Mbit / s	79,61 MHz
Grøstl-384/512	documento http://eprint.iacr.org	Funcionalidad básica	P & Q paralelo permutación, S-caja en LUT	Xilinx Virtex 5	19.161 rebanadas	6090 Mbit / s	83,33 MHz
Grøstl-384/512	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo	P & Q permutación en paralelo	Xilinx Virtex 5	5419 rebanadas	15 395 Mbit / s	210.5 MHz
Grøstl-384/512	documento http://eprint.iacr.org	Totalmente autónomo	Compartidas P & Q permutación	Xilinx Spartan 3	8308 rebanadas	3474 Mbit / s	95 MHz
Grøstl-512	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	4845 rebanadas	3619 Mbit / s	123.4 MHz
Grøstl-512	documento http://eprint.iacr.org	Totalmente autónomo	permutaciones P & Q	Xilinx Virtex	3138	10 314 Mbit /	292.1 MHz

			intercalados	5	rebanadas	s	
Grøstl-512	documento http://eprint.iacr.org	Totalmente autónomo	permutaciones P & Q intercalados	Altera Stratix III	12355 ALUTs	7142 Mbit / s	202.3 MHz
JH-256	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	1018 rebanadas	5416 Mbit / s	380.8 MHz
JH-256	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Altera Stratix III	3525 ALUTs	5515 Mbit / s	387.8 MHz
JH-256	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	2661 rebanadas	2639 Mbit / s	201 MHz
JH	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	1291 rebanadas	1941 Mbit / s	250,13 MHz
JH-512	documento http://eprint.iacr.org	Totalmente autónomo		Xilinx Virtex 5	1104 rebanadas	5610 Mbit / s	394.5 MHz
JH-512	documento http://eprint.iacr.org	Totalmente autónomo		Altera Stratix III	3709 ALUTs	5556 Mbit / s	390.6 MHz
Keccak	Actualización de especificaciones. (v1.2) http://keccak.noekeon.org/	Totalmente autónomo	Núcleo (función redonda, registro de estado) y de E / S de	Altera el ciclón III	5776 GE	7500 Mbit / s	133 MHz

			búfer				
Keccak	Actualización de especificaciones. (v1.2) http://keccak.noekeon.org/	Totalmente autónomo	Núcleo (función redonda, registro de estado) y de E / S de búfer	Altera Stratix III	4713 ALUTs	12 400 Mbit / s	218 MHz
Keccak	http://keccak.noekeon.org/	Totalmente autónomo	Núcleo (función redonda, registro de estado) sólo	Xilinx Spartan 3A	3393 rebanadas	4800 Mbit / s	85 MHz
Keccak	Actualización de especificaciones. (v1.2) http://keccak.noekeon.org/	Totalmente autónomo	Núcleo (función redonda, registro de estado) y de E / S de búfer	Xilinx Virtex 5	1412 rebanadas	6900 Mbit / s	122 MHz
Keccak (-224)	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	1117 rebanadas	5915 Mbit / s	189 MHz
Keccak (-256)	documento http://eprint.iacr.org	Totalmente autónomo		Xilinx Virtex 5	1272 rebanadas	12 817 Mbit / s	282.7 MHz
Keccak (-256)	documento http://eprint.iacr.org	Totalmente autónomo		Altera Stratix III	4213 ALUTs	12 393 Mbit / s	273.4 MHz
Keccak (-256)	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	1117 rebanadas	6263 Mbit / s	189 MHz

Keccak (-256)	http://www.ucc.ie/en/crypto/SHA-3Hardware/	Totalmente autónomo		Xilinx Virtex 5	1433 rebanadas	8397 Mbit / s	205 MHz
Keccak (-384)	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	1117 rebanadas	8190 Mbit / s	189 MHz
Keccak (-512)	http://eprint.iacr.org/2010/445.pdf	Totalmente autónomo		Xilinx Virtex 5	1117 rebanadas	8518 Mbit / s	189 MHz
Keccak (-512)	documento http://eprint.iacr.org	Totalmente autónomo		Xilinx Virtex 5	1257 rebanadas	6845 Mbit / s	285.2 MHz
Keccak (-512)	documento http://eprint.iacr.org	Totalmente autónomo		Altera Stratix III	3979 ALUTs	7310 Mbit / s	304.6 MHz
Keccak	documentos aportados al NIST	Funcionalidad básica	Una ronda Keccak por ciclo	Xilinx Spartan 3	2024 rebanadas	3460 Mbit / s	81.4 MHz
Keccak	documentos aportados al NIST	Funcionalidad básica	Una ronda Keccak por ciclo	Xilinx Virtex-II	2024 rebanadas	5810 Mbit / s	136.6 MHz
Keccak	documentos aportados al NIST	Funcionalidad básica	Una ronda Keccak por ciclo	Xilinx Virtex 4	2024 rebanadas	6070 Mbit / s	142.9 MHz
Madeja-256-h	http://www.skein-hash.info/sites/default/files/skein_fpga.pdf	Funcionalidad básica	UBI componente	Xilinx Virtex 5	1001 rebanadas	408,7 Mbit / s	114.9 MHz

Madeja, 256-256	http://eprint.iacr.org/2009/159.pdf	Totalmente autónomo	8 rondas Threefish desenrolló	Xilinx Virtex 5	937 rebanadas	1751 Mbit / s	68.4 MHz
Madeja, 256-256	http://eprint.iacr.org/2009/159.pdf	Totalmente autónomo	8 rondas Threefish desenrolló	Xilinx Spartan 3	2421 rebanadas	669 Mbit / s	26,14 MHz
Madeja, 256-256	http://eprint.iacr.org/2010/010.pdf	Totalmente autónomo		Xilinx Virtex 5	854 rebanadas	1482 Mbit / s	115 MHz
Madeja, 512-256	documento http://eprint.iacr.org	Totalmente autónomo	4 rondas Threefish desenrolló	Xilinx Virtex 5	1621 rebanadas	3178 Mbit / s	118.0 MHz
Madeja, 512-256	documento http://eprint.iacr.org	Totalmente autónomo	4 rondas Threefish desenrolló	Altera Stratix III	4645 ALUTs	2503 Mbit / s	92.9 MHz
Madeja, 256-256	http://eprint.iacr.org/2010/010.pdf	Totalmente autónomo		Xilinx Virtex 5	854 rebanadas	1402 Mbit / s	115 MHz
Madeja-512-h	documentos aportados al NIST	Funcionalidad básica	UBI componente	Xilinx Virtex 5	1877 rebanadas	817,4 Mbit / s	114.9 MHz
Madeja-512 a 512	documento http://eprint.iacr.org	Totalmente autónomo	8 rondas Threefish desenrolló	Xilinx Virtex 5	1632 rebanadas	3535 Mbit / s	69,04 MHz
Madeja-512	documento http://eprint.iacr.org	Totalmente autónomo	8 rondas Threefish desenrolló	Xilinx	4273	1365 Mbit / s	26,66 MHz

a 512				Spartan 3	rebanadas		
Madeja-512	http://www.ucc.ie/en/crypto/SHA-3Hardware/	Totalmente autónomo		Xilinx Virtex 5	1786 rebanadas	1945 Mbit / s	83,65 MHz
Madeja-512 a 512	documento http://eprint.iacr.org	Totalmente autónomo	4 rondas Threefish desenrolló	Xilinx Virtex 5	1716 rebanadas	3209 Mbit / s	119.1 MHz
Madeja-512 a 512	documento http://eprint.iacr.org	Totalmente autónomo	4 rondas Threefish desenrolló	Altera Stratix III	4794 ALUTs	2434 Mbit / s	90.3 MHz

AREA BAJO IMPLEMENTACIONES (FPGA)

Nombre de la función hash	Referencia / HDL	Impl. Ámbito de aplicación	Detalles de la implementación	Tecnología	Tamaño	Rendimiento	Frecuencia de reloj
BLAKE-32	http://eprint.iacr.org/2010/173.pdf	Totalmente autónomo	Reprogramada la función G	Xilinx Spartan-3	124 rebanadas	115 Mbit / s	190.0 MHz
BLAKE-32	http://eprint.iacr.org/2010/173.pdf	Totalmente autónomo	Reprogramada la función G	Xilinx Virtex-4	124 rebanadas	216 Mbit / s	357.0 MHz
BLAKE-32	http://eprint.iacr.org/2010/173.pdf	Totalmente	Reprogramada la función G	Xilinx Virtex-5	56 rebanadas	225 Mbit / s	372.0 MHz

		autónomo					
BLAKE-32	http://eprint.iacr.org/2010/173.pdf	Totalmente autónomo	Reprogramada la función G	Altera el ciclón III	285 GE	116 Mbit / s	192.0 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con una unidad de la función G	Xilinx Virtex-II Pro	958 rebanadas	371 Mbit / s	59.0 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con una unidad de la función G	Xilinx Virtex 4	960 rebanadas	430 Mbit / s	68.0 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con una unidad de la función G	Xilinx Virtex 5	390 rebanadas	575 Mbit / s	91.0 MHz
BLAKE-64	documento http://eprint.iacr.org	Totalmente autónomo	Reprogramada la función G	Xilinx Spartan-3	229 rebanadas	138 Mbit / s	158.0 MHz
BLAKE-64	documento http://eprint.iacr.org	Totalmente autónomo	Reprogramada la función G	Xilinx Virtex-4	230 rebanadas	219 Mbit / s	250.0 MHz
BLAKE-64	documento http://eprint.iacr.org	Totalmente autónomo	Reprogramada la función G	Xilinx Virtex-5	108 rebanadas	314 Mbit / s	358.0 MHz
BLAKE-64	documento http://eprint.iacr.org	Totalmente autónomo	Reprogramada la función G	Altera el ciclón III	542 GE	123 Mbit / s	140.0 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con	Xilinx Virtex-II	1802	326 Mbit / s	36.0 MHz

			una unidad de la función G	Pro	rebanadas		
BLAKE-64	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con una unidad de la función G	Xilinx Virtex 4	1856 rebanadas	381 Mbit / s	42.0 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con una unidad de la función G	Xilinx Virtex 5	939 rebanadas	533 Mbit / s	59.0 MHz
Grøstl-224/256	documento http://eprint.iacr.org	Totalmente autónomo	camino de datos de 64 bits, P y Q de permutación en paralelo	Xilinx Spartan 3	2486 rebanadas	404 Mbit / s	63.2 MHz
Grøstl-224/256	documento http://eprint.iacr.org	Totalmente autónomo	camino de datos de 64 bits, P y Q de permutación en paralelo	Xilinx Virtex 2 Pro	2754 rebanadas	512 Mbit / s	81.5 MHz
Grøstl-224/256	documento http://eprint.iacr.org	Totalmente autónomo	Compartidas P & permutación Q, S-Box sobre la base de la aritmética campo compuesto	Xilinx Spartan 3	1276 rebanadas	192 Mbit / s	60 MHz
Grøstl-384/512	documento http://eprint.iacr.org	Totalmente autónomo	Compartidas P & permutación Q, S-Box sobre la base de la aritmética campo compuesto	Xilinx Spartan 3	2110 rebanadas	144 Mbit / s	63 MHz
Keccak	Actualización de especificaciones. (v1.2) http://keccak.noekeon.org/	Uso de la memoria externa	pequeño núcleo con la memoria del sistema	Altera Stratix III	855 ALUTs	96.8 Mbit / s	366 MHz
Keccak	Actualización de especificaciones.	Uso de la memoria	pequeño núcleo con la memoria	Altera el ciclón III	1559 GE	47.8 Mbit / s	181 MHz

	(v1.2) http://keccak.noekeon.org/	externa	del sistema				
Keccak	Actualización de especificaciones. (v1.2) http://keccak.noekeon.org/	Uso de la memoria externa	pequeño núcleo con la memoria del sistema	Xilinx Virtex 5	444 rebanadas	70.1 Mbit / s	265 MHz
Madeja, 256-256	documento http://eprint.iacr.org	Funcionalidad básica	Una ronda de Threefish iterado	Altera Stratix III	1385 ALUTs	573,9 Mbit / s	161,42 MHz

IMPLEMENTACIONES DE ALTA VELOCIDAD (ASIC)

Nombre de la función hash	**Referencia / HDL**	**Impl. Ámbito de aplicación**	**Detalles de la implementación**	**Tecnología**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 8 unidades de la función G	UMC 0,18 micras	58.30 kGates	5295 Mbit / s	114 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 4 unidades de la función G	UMC 0,18 micras	41.31 kGates	4153 Mbit / s	170 MHz
BLAKE-32	documento http://eprint.iacr.org	Funcionalidad básica	Compresión de la función con 8 unidades de la función G e I / O	STM 90 nm	53 kGates	4475 Mbit / s (*)	96,15 MHz

			registros				
BLAKE-32	documento http://eprint.iacr.org	Totalmente autónomo	Compresión de la función con 4 unidades de la función G con CSA	UMC 0,18 micras	45.64 kGates	3971 Mbit / s	170,64 MHz
BLAKE-32	documento http://eprint.iacr.org	Totalmente autónomo	Cuatro funciones paralelas G módulos	UMC 90 nm	47.5 kGates	9752 Mbit / s	400 MHz
BLAKE-32	página web TV	Totalmente autónomo		UMC 0,13 micras	43.52 kGates	4645 Mbit / s	200 MHz
BLAKE-32	RCIS página web	Totalmente autónomo		STM 90 nm	37 kGates	6668 Mbit / s	286.5 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 8 unidades de la función G	UMC 0,18 micras	79 kGates	6376 Mbit / s	137 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 4 unidades de la función G	UMC 0,18 micras	48 kGates	5847 Mbit / s	240 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 8 unidades de la función G	UMC 0,13 micras	67 kGates	9365 Mbit / s	201 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 4 unidades de la función G	UMC 0,13 micras	43 kGates	8047 Mbit / s	330 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Totalmente	Compresión de la función con 8	UMC 90 nm	65 kGates	17 498 Mbit / s	376 MHz

		autónomo	unidades de la función G				
BLAKE-32	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 4 unidades de la función G	UMC 90 nm	38 kGates	15 143 Mbit / s	621 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 8 unidades de la función G	UMC 0,18 micras	132,47 kGates	5910 Mbit / s	87 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con 4 unidades de la función G	UMC 0,18 micras	82.73 kGates	4810 Mbit / s	136 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 8 unidades de la función G	UMC 0,18 micras	147 kGates	7216 Mbit / s	106 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 4 unidades de la función G	UMC 0,18 micras	98 kGates	7192 Mbit / s	204 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 8 unidades de la función G	UMC 0,13 micras	139 kGates	10 802 Mbit / s	158 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 4 unidades de la función G	UMC 0,13 micras	92 kGates	10 265 Mbit / s	291 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Totalmente autónomo	Compresión de la función con 8 unidades de la función G	UMC 90 nm	128 kGates	20 317 Mbit / s	298 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Totalmente	Compresión de la función con 4	UMC 90 nm	79 kGates	18 782 Mbit / s	532 MHz

		autónomo	unidades de la función G				
Grøstl-256	documento http://eprint.iacr.org	Totalmente autónomo	Una permutación compartida para P & Q, una etapa de la tubería	UMC 0,18 micras	58.40 kGates	6290 Mbit / s	270,27 MHz
Grøstl-256	documento http://eprint.iacr.org	Totalmente autónomo	permutación P y Q intercalados con una etapa de la tubería, caja-S como LUT	UMC 90 nm	135 kGates	16 254 Mbit / s	667 MHz
Grøstl-256	página web TV	Totalmente autónomo		UMC 0,13 micras	110,11 kGates	9606 Mbit / s	188 MHz
Grøstl-256	http://staff.aist.go.jp	Totalmente autónomo		STM 90 nm	139.1 kGates	17 297 Mbit / s	337.8 MHz
Grøstl-256	http://staff.aist.go.jp	Totalmente autónomo		STM 90 nm	120.8 kGates	16 275 Mbit / s	349.7 MHz
Grøstl-384/512	http://www.groestl.info/Groestl.pdf	Totalmente autónomo	P & Q permutación en paralelo	UMC 0,18 micras	341 kGates	6225 Mbit / s	85.1 MHz
JH-256	documento http://eprint.iacr.org	Totalmente autónomo	320 cajas-S, una ronda de R_8 por ciclo	UMC 0,18 micras	58.83 kGates	4991 Mbit / s	380,22 MHz
JH-256	http://www.iis.ee.ethz.ch/~sha3/	Totalmente autónomo	S-cajas como LUT, constantes almacenadas	UMC 90 nm	80 kGates	10 807 Mbit / s	760 MHz

JH-256	http://www.iis.ee.ethz.ch/~sha3/	Totalmente autónomo		UMC 0,13 micras	62.42 kGates	5128 Mbit / s	391 MHz
JH-256	http://staff.aist.go.jp	Totalmente autónomo		STM 90 nm	54.6 kGates	10 022 Mbit / s	763.4 MHz
Keccak	Actualización de especificaciones. (V1.2) http://keccak.noekeon.org/	Totalmente autónomo	Núcleo (función redonda, registro de estado) y de E / S de búfer	ST 0.13 micras	48 kGates	29 900 Mbit / s	526 MHz
Keccak	http://keccak.noekeon.org/	Totalmente autónomo	Núcleo (función redonda, registro de estado) sólo	ST 0.13 micras	40 kGates	15 000 Mbit / s	500 MHz
Keccak (-256)	documento http://eprint.iacr.org	Totalmente autónomo	Un ejemplo de Keccak-f ronda	UMC 0,18 micras	56.32 kGates	21 229 Mbit / s	487,80 MHz
Keccak (-256)	http://www.iis.ee.ethz.ch/~sha3/	Totalmente autónomo	Una ronda por ciclo	UMC 90 nm	50 kGates	43 011 Mbit / s	949 MHz
Keccak (-256)	http://www.iis.ee.ethz.ch/~sha3/	Totalmente autónomo		UMC 0,13 micras	47.43 kGates	15 457 Mbit / s	377 MHz
Keccak	documento http://eprint.iacr.org	Funcionalidad básica	Una Keccak-f ronda por ciclo	Synopsys 90 nm	10.5 kGates	19 320 Mbit / s	454.5 MHz
Keccak (-256)	http://www.iis.ee.ethz.ch/~sha3/	Totalmente autónomo		STM 90 nm	50.7 kGates	33 333 Mbit / s	781.3 MHz

Keccak (-256)	http://www.iis.ee.ethz.ch/~sha3/	Totalmente autónomo		STM 90 nm	55.9 kGates	43 986 Mbit / s	1030.9 MHz
Madeja, 256-256	documento http://eprint.iacr.org	Totalmente autónomo	8 rondas Threefish desenrolló	UMC 0,18 micras	53.87 kGates	1762 Mbit / s	68.8 MHz
Madeja, 256-256	documento http://eprint.iacr.org	Funcionalidad básica	Las 72 rondas Threefish desenrolló	STM 90 nm	369 kGates	3126 Mbit / s (*)	12,21 MHz
Madeja, 256-256	documento http://eprint.iacr.org	Totalmente autónomo	8 rondas Threefish desenrolló	UMC 0,18 micras	58.61 kGates	1882 Mbit / s	73,52 MHz
Madeja, 256-256	http://www.iis.ee.ethz.ch/~sha3/	Totalmente autónomo	Cuatro rondas desenrolló Threefish	UMC 90 nm	50 kGates	3558 Mbit / s	264 MHz
Madeja, 256-256	http://www.iis.ee.ethz.ch/~sha3/	Totalmente autónomo		UMC 0,13 micras	40.9 kGates	1941 Mbit / s	159 MHz
Madeja, 256-256	http://staff.aist.go.jp	Totalmente autónomo		STM 90 nm	43.1 kGates	3295 Mbit / s	270.3 MHz
Madeja-512 a 512	documento http://eprint.iacr.org	Totalmente autónomo	8 rondas Threefish desenrolló	UMC 0,18 micras	102,04 kGates	2502 Mbit / s	48,87 MHz
Madeja-512	documento http://eprint.iacr.org	Totalmente autónomo	8 rondas Threefish desenrolló	Intel de 32 nm	57.93 kGates	32 320 Mbit / s	631.31 MHz

(*) Rendimiento máximo estimado para la demora mínima de la función de compresión: 1000 * (Tamaño de entrada en bits) / [(compresión de la función de retardo en ns) * (Número de ciclos)] = Rendimiento en Mbit / s.

AREA BAJO IMPLEMENTACIONES (ASIC)

Nombre de la función hash	Referencia / HDL	Impl. Ámbito de aplicación	Detalles de la implementación	Tecnología	Tamaño	Rendimiento	Frecuencia de reloj
BLAKE-32	documento http://eprint.iacr.org	Totalmente autónomo	Una función G en 11 ciclos	AMS 0.35 micras	25.57 kGates	15.4 Mbit / s	31,25 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con una sola unidad de la función G	UMC 0,18 micras	10.54 kGates	253 Mbit / s	40 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con una unidad de la función media G	UMC 0,18 micras	9.89 kGates	127 Mbit / s	40 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Totalmente autónomo	Una serpiente y 4 de la palabra-cierre matriz	UMC 0,18 micras	13.56 kGates	135 Mbit / s	215 MHz
BLAKE-32	http://131002.net/blake/blake.pdf	Uso de la memoria externa	Una serpiente y 4 de la palabra-cierre matriz	UMC 0,18 micras	8.60 kGates	62 Mbit / s	100 MHz

BLAKE-64	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con una sola unidad de la función G	UMC 0,18 micras	20.61 kGates	181 Mbit / s	20 MHz
BLAKE-64	http://131002.net/blake/blake.pdf	Funcionalidad básica	Compresión de la función con una unidad de la función media G	UMC 0,18 micras	19.46 kGates	91 Mbit / s	20 MHz
Grøstl-224/256	documento http://eprint.iacr.org	Totalmente autónomo	Camino de datos de 64 bits, permutación P & Q compartida	AMS 0.35 micras	14.62 kGates	145,9 Mbit / s	55,87 MHz
Grøstl-224/256	http://www.groestl.info/Groestl.pdf	Totalmente autónomo	Camino de datos de 64 bits, permutación P & Q compartida	UMC 0,18 micras	17 kGates	645 Mbit / s	246.9 MHz
Grøstl-256	http://staff.aist.go.jp	Totalmente autónomo		STM 90 nm	34.8 kGates	2478 Mbit / s	101.6 MHz
Keccak	Actualización de especificaciones. (V1.2) http://keccak.noekeon.org/	Uso de la memoria externa	pequeño núcleo con la memoria del sistema	ST 0.13 micras	6.5 kGates	176,4 Mbit / s (*)	666.7 MHz
Keccak	Actualización de especificaciones. (V1.2) http://keccak.noekeon.org/	Uso de la memoria externa	pequeño núcleo con la memoria del sistema, frecuencia de reloj. limitado a 200 MHz	ST 0.13 micras	5 kGates	52.9 Mbit / s (**)	200 MHz
Madeja, 256-256	documento http://eprint.iacr.org	Totalmente autónomo	Camino de datos de 64 bits	AMS 0.35 micras	12.89 kGates	19.8 Mbit / s	80 MHz
Madeja,	documento http://eprint.iacr.org	Funcionalidad	Una ronda de Threefish iterado	STM 90 nm	21 kGates	1.018,8 Mbit / s	286,53

256-256		básica				(***)	MHz

(*) Estimación para la interfaz de memoria de 64 bits: (1024 bits / permutación) * (666.7 * 10 ^ 6 ciclos / s) / (3870 ciclos / permutación) = 176.41 * 10 ^ 6 bits / s

(**) Estimación para la interfaz de memoria de 64 bits: (1024 bits / permutación) * (200 * 10 ^ 6 ciclos / s) / (3870 ciclos / permutación) = 52.92 * 10 ^ 6 bits / s

(***) Rendimiento máximo estimado para la demora mínima de la función de compresión: 1000 * (Tamaño de entrada en bits) / [(compresión de la función de retardo en ns) * (Número de ciclos)] = Rendimiento en Mbit / s

ESTUDIOS COMPARATIVOS CON CONFIGURACIONES SIMILARES

Referencia	**HDL**	**Categoría**	**Impl. Ámbito de aplicación**	**Tecnología**
documento http://eprint.iacr.org	N / A	FPGA de alta velocidad	Funcionalidad básica	Altera Stratix III
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32	Compresión de la función con 8 unidades de la función G e I / O registros	5435 ALUTs	2.186,2 Mbit / s	46,97 MHz
Madeja, 256-256	Las 72 rondas Threefish desenrollado (dispositivo demasiado pequeño)	N / A	N / A	N / A

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
documento http://eprint.iacr.org	N / A	ASIC de alta velocidad	Funcionalidad básica	STM 90 nm
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32	Compresión de la función con 8 unidades de la función G e I / O registros	53 kGates	4475 Mbit / s (*)	96,15 MHz
Madeja, 256-256	Las 72 rondas Threefish desenrolló	369 kGates	3126 Mbit / s (*)	12,21 MHz

(*) Rendimiento máximo estimado para la demora mínima de la función de compresión: 1000 * (Tamaño de entrada en bits) / [(compresión de la función de retardo en ns) * (Número de ciclos)] = Rendimiento en Mbit / s.

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
documento http://eprint.iacr.org	RCIS página web	FPGA de alta velocidad	Totalmente autónomo	Xilinx Virtex 5

Nombre de la función hash	Impl. Detalles	Tamaño	Rendimiento	Frecuencia de reloj
BLAKE-32		1660 rebanadas	2676 Mbit / s	115 MHz
Grøstl-256		4057 rebanadas	5171 Mbit / s	101 MHz
Madeja-256		854 rebanadas	1482 Mbit / s	115 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
documento http://eprint.iacr.org	N / A	FPGA de alta velocidad	Funcionalidad básica	Xilinx Spartan 3
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
Grøstl-224/256	P & Q permutación de forma paralela, S-caja en BRAM	4827 rebanadas	3660 Mbit / s	71,53 MHz
Grøstl-384/512	P & Q paralelo permutación, S-caja en LUT	17.452 rebanadas	3180 Mbit / s	79,61 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
documento http://eprint.iacr.org	N / A	FPGA de alta velocidad	Funcionalidad básica	Xilinx Virtex 5
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
Grøstl-224/256	P & Q permutación de forma paralela, S-caja en BRAM	4516 rebanadas	7310 Mbit / s	142,87 MHz
Grøstl-384/512	P & Q paralelo permutación, S-caja en LUT	19.161 rebanadas	6090 Mbit / s	83,33 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
documento http://eprint.iacr.org		ASIC de alta velocidad	Totalmente autónomo	UMC 0,18 micras

Nombre de la función hash	Impl. Detalles	Tamaño	Rendimiento	Frecuencia de reloj
BLAKE-32	Compresión de la función con 4 unidades de la función G con CSA	45.64 kGates	3971 Mbit / s	170,64 MHz
Grøstl-256	Una permutación compartida para P & Q, una etapa de la tubería	58.40 kGates	6290 Mbit / s	270,27 MHz
Hamsi-256	Tres casos de P / función Pf desenrolló	58.66 kGates	5565 Mbit / s	173,91 MHz
JH-256	320 cajas-S, una ronda de R_8 por ciclo	58.83 kGates	4991 Mbit / s	380,22 MHz
Keccak (-256)	Un ejemplo de Keccak-f ronda	56.32 kGates	21 229 Mbit / s	487,80 MHz
Madeja, 256-256	8 rondas Threefish desenrolló	58.61 kGates	1882 Mbit / s	73,52 MHz
Madeja-512 a 512	8 rondas Threefish desenrolló	102,04 kGates	2502 Mbit / s	48,87 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
documento http://eprint.iacr.org	N / A	Bajo la zona del ASIC	Totalmente autónomo	AMS 0.35 micras
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32	Una función G en 11 ciclos	25.57 kGates	15.4 Mbit / s	31,25 MHz
Grøstl-224/256	Camino de datos de 64 bits, permutación P & Q compartida	14.62 kGates	145,9 Mbit / s	55,87 MHz
Madeja, 256-256	Camino de datos de 64 bits	12.89 kGates	19.8 Mbit / s	80 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
documento http://eprint.iacr.org	ETH página web	ASIC de alta velocidad	Totalmente autónomo	UMC 90 nm
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32	Cuatro funciones paralelas G módulos	47.5 kGates	9752 Mbit / s	400 MHz
Grøstl-256	permutación P y Q intercalados con una etapa de la tubería, caja-S como LUT	135 kGates	16 254 Mbit / s	667 MHz
JH-256	S-cajas como LUT, constantes almacenadas	80 kGates	10 807 Mbit / s	760 MHz
Keccak (-256)	Una ronda por ciclo	50 kGates	43 011 Mbit / s	949 MHz
Madeja, 256-256	Cuatro rondas desenrolló Threefish	50 kGates	3558 Mbit / s	264 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
documento http://eprint.iacr.org		High-speed FPGA	Totalmente autónomo	Xilinx Virtex 5
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32	4 G unidades de la función por iteración	1523 rebanadas	3143 Mbit / s	128.9 MHz
BLAKE-64	4 G unidades de la función por iteración	3064 rebanadas	3520 Mbit / s	99.7 MHz
Grøstl-256	permutaciones P & Q intercalados	1597 rebanadas	7885 Mbit / s	323.4 MHz
Grøstl-512	permutaciones P & Q intercalados	3138 rebanadas	10 314 Mbit / s	292.1 MHz
JH-256		1018 rebanadas	5416 Mbit / s	380.8 MHz
JH-512		1104 rebanadas	5610 Mbit / s	394.5 MHz
Keccak(-256)		1272 slices	12817 Mbit/s	282.7 MHz
Keccak(-512)		1257 slices	6845 Mbit/s	285.2 MHz
Skein-512-256	4 Threefish rounds unrolled	1621 slices	3178 Mbit/s	118.0 MHz

Skein-512-512	4 Threefish rounds unrolled	1716 slices	3209 Mbit/s	119.1 MHz

Referencia	**HDL**	**Categoría**	**Impl. Ámbito de aplicación**	**Tecnología**
documento http://eprint.iacr.org	N / A	High-speed FPGA	Totalmente autónomo	Altera Stratix III
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32	4 G unidades de la función por iteración	3635 ALUTs	2901 Mbit / s	119.0 MHz
BLAKE-64	4 G unidades de la función por iteración	7086 ALUTs	3161 Mbit / s	89.5 MHz
Grøstl-256	permutaciones P & Q intercalados	6350 ALUTs	5380 Mbit / s	220.7 MHz
Grøstl-512	permutaciones P & Q intercalados	12355 ALUTs	7142 Mbit / s	202.3 MHz
JH-256		3525 ALUTs	5515 Mbit / s	387.8 MHz
JH-512		3709 ALUTs	5556 Mbit / s	390.6 MHz
Keccak(-256)		4213 ALUTs	12393 Mbit/s	273.4 MHz

Keccak(-512)		3979 ALUTs	7310 Mbit / s	304.6 MHz
Skein-512-256	4 Threefish rounds unrolled	4645 ALUTs	2503 Mbit/s	92.9 MHz
Skein-512-512	4 Threefish rounds unrolled	4794 ALUTs	2434 Mbit/s	90.3 MHz

Referencia	**HDL**	**Categoría**	**Impl. Ámbito de aplicación**	**Tecnología**
http://csrc.nist.gov/	UCC página web	FPGA de alta velocidad	Totalmente autónomo	Xilinx Virtex 5
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32		1118 rebanadas	1169 Mbit / s	118,06 MHz
BLAKE-64		1718 rebanadas	1299 Mbit / s	90,91 MHz
Grøstl-256		2391 rebanadas	3242 Mbit / s	101,32 MHz
Grøstl-512		4845 rebanadas	3619 Mbit / s	123.4 MHz
JH		1291 rebanadas	1941 Mbit / s	250,13 MHz
Keccak (-224)		1117 rebanadas	5915 Mbit / s	189 MHz

Keccak (-256)		1117 rebanadas	6263 Mbit / s	189 MHz
Keccak (-384)		1117 rebanadas	8190 Mbit / s	189 MHz
Keccak (-512)		1117 rebanadas	8518 Mbit / s	189 MHz
Madeja-512		1786 rebanadas	1945 Mbit / s	83,65 MHz

Referencia	**HDL**	**Categoría**	**Impl. Ámbito de aplicación**	**Tecnología**
http://csrc.nist.gov/	RCIS página web	FPGA de alta velocidad	Totalmente autónomo	Xilinx Virtex 5
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32		1660 rebanadas	2676 Mbit / s	115 MHz
Grøstl-256		2616 rebanadas	7885 Mbit / s	154 MHz
JH-256		2661 rebanadas	2639 Mbit / s	201 MHz
Keccak (-256)		1433 rebanadas	8397 Mbit / s	205 MHz
Madeja, 256-256		854 rebanadas	1402 Mbit / s	115 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
ttp://staff.aist.go.jp/	RCIS página web	ASIC de alta velocidad	Totalmente autónomo	STM 90 nm
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32		37 kGates	6668 Mbit / s	286.5 MHz
Grøstl-256		139.1 kGates	17 297 Mbit / s	337.8 MHz
JH-256		54.6 kGates	10 022 Mbit / s	763.4 MHz
Keccak (-256)		50.7 kGates	33 333 Mbit / s	781.3 MHz
Madeja, 256-256		43.1 kGates	3295 Mbit / s	270.3 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
http://csrc.nist.gov/		FPGA de alta velocidad	Funcionalidad básica	Xilinx Spartan 3

Nombre de la función hash	Impl. Detalles	Tamaño	Rendimiento	Frecuencia de reloj
Keccak	Una Keccak-f ronda por ciclo	2024 rebanadas	3460 Mbit / s	81.4 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
http://csrc.nist.gov/	N / A	FPGA de alta velocidad	Funcionalidad básica	Xilinx Virtex-II
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
Keccak	Una Keccak-f ronda por ciclo	2024 rebanadas	5810 Mbit / s	136.6 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
http://csrc.nist.gov/	N / A	FPGA de alta velocidad	Funcionalidad básica	Xilinx Virtex 4
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
Keccak	Una Keccak-f ronda por ciclo	2024 rebanadas	6070 Mbit / s	142.9 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
http://csrc.nist.gov/	N / A	ASIC de alta velocidad	Funcionalidad básica	Synopsys 90 nm
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
Keccak	Una Keccak-f ronda por ciclo	10.5 kGates	19 320 Mbit / s	454.5 MHz

Referencia	HDL	Categoría	Impl. Ámbito de aplicación	Tecnología
http://csrc.nist.gov/	VT página web	ASIC de alta velocidad	Totalmente autónomo	UMC 0,13 micras
Nombre de la función hash	**Impl. Detalles**	**Tamaño**	**Rendimiento**	**Frecuencia de reloj**
BLAKE-32		43.52 kGates	4645 Mbit / s	200 MHz
Grøstl-256		110,11 kGates	9606 Mbit / s	188 MHz

JH-256		62.42 kGates	5128 Mbit / s	391 MHz
Keccak (-256)		47.43 kGates	15 457 Mbit / s	377 MHz
Madeja, 256-256		40.9 kGates	1941 Mbit / s	159 MHz